



Accuracer Developer's Guide

(c) AidAim Software, 2000-2009

Place your own product logo here and modify the layout of your print manual/PDF:
In Help & Manual, click "Tools" > "Print Manual Designer" and open this manual template to edit it.

Title page 1

Use this page to introduce the product

by

This is "Title Page 1" - you may use this page to introduce your product, show title, author, copyright, company logos, etc.

This page intentionally starts on an odd page, so that it is on the right half of an open book from the readers point of view. This is the reason why the previous page was blank (the previous page is the back side of the cover)

Accuracer Developer's Guide

(c) AidAim Software, 2000-2009

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: August 2009 in (whereever you are located)

Publisher

...enter name...

Managing Editor

...enter name...

Technical Editors

...enter name...

...enter name...

Cover Designer

...enter name...

Team Coordinator

...enter name...

Production

...enter name...

Special thanks to:

All the people who contributed to this document, to mum and dad and grandpa, to my sisters and brothers and mothers in law, to our secretary Kathrin, to the graphic artist who created this great product logo on the cover page (sorry, don't remember your name at the moment but you did a great work), to the pizza service down the street (your daily Capricciosas saved our lives), to the copy shop where this document will be duplicated, and and and...

Last not least, we want to thank EC Software who wrote this great help tool called HELP & MANUAL which printed this document.

Table of Contents

Foreword	I
Part I Guía de Desarrollo Accuracer	2
1 Introduction	2
2 Características	2
3 Solicitando ayuda y soporte técnico	5
4 How to Buy	5
5 Trabajando con tablas	6
Creando un archivo de base de datos	6
Configurando los componentes de tabla y base de datos	6
Creando tablas	6
Abriendo y cerrando tablas	8
Navegando las tablas	9
Filtrando registros	10
Buscando registros	11
Clasificando registros	13
Uso de campos BLOB	14
Campos Varchar y BLOB	15
6 Operaciones Avanzadas con Tablas	17
Resestructurando una tabla	17
7 Referencia SQL	18
Descripción	18
Convención de nombres	19
Usando parámetros	25
Operadores	26
HEX constants	28
Funciones	28
Funciones agregadas.....	29
Función AVG.....	29
Función COUNT.....	29
GROUP_CONCAT Function.....	30
Función MIN.....	30
Función MAX.....	31
Función SUM.....	31
Funciones de Fecha y Hora.....	31
Función CURRENT_DATE.....	32
Función CURRENT_TIME.....	32
Funciones CURRENT_TIMESTAMP, NOW AND SYSDATE.....	33
Función DAY.....	33
Función DAYNAME.....	33
Función DAYOFWEEK.....	33
Función EXTRACT.....	34
Función HOUR.....	34
Función MINUTE.....	35
Función MONTH.....	35
Función MONTHNAME.....	35
Función MSECOND.....	35

Función QUARTER.....	36
Función SECOND.....	36
Función TODATE.....	36
Función TOSTRING.....	38
Función WEEKDAY.....	39
Función YEAR.....	39
Funciones Miscelaneas.....	39
Función ISNULL.....	40
Función LASTAUTOINC.....	40
Mathematical Functions.....	40
ABS Function.....	41
SIGN Function.....	41
MOD Function.....	42
FLOOR Function.....	42
CEILING Function.....	42
CUMSUM Function.....	42
CUMPROD Function.....	43
ROUND Function.....	43
TRUNCATE Function.....	43
POWER Function.....	44
HEX Function.....	44
RANDOM Function.....	44
Funciones String.....	45
Función LENGTH.....	45
Función LOWER.....	46
Función LTRIM.....	46
Función POS.....	46
Función RTRIM.....	47
Función SUBSTRING.....	47
Función TRIM.....	47
Función UPPER.....	47
Funciones de conversión de tipos.....	48
Función CAST.....	48
Función TOBLOB.....	49
Declaración SELECT.....	49
Declaración INSERT.....	54
Declaración UPDATE.....	54
Declaración DELETE.....	55
CREATE DATABASE Statement.....	56
DROP DATABASE Statement.....	56
Declaración CREATE TABLE.....	57
Declaración ALTER TABLE.....	59
Declaración DROP TABLE.....	61
Declaración CREATE INDEX.....	61
Declaración DROP INDEX.....	62
Declaración START TRANSACTION.....	62
Declaración COMMIT.....	63
Declaración ROLLBACK.....	63
8 Mecanismo de bloqueo y transacciones, Multi-Usuario y Multi-Tarea.....	64
Soporte Multi-Usuario y Multi-Tarea.....	64
Mecanismo de Bloqueo.....	65
Transacciones.....	67
9 Motor Cliente-Servidor.....	69
Introducción.....	69

10 Migración	71
Sobrevista	71
Migración desde BDE	72
Migración desde EasyTable	72
Migración desde otros sistemas de base de datos y plataformas	72
Importar y Exportar	73
11 Afinamiento y optimización	74
Revisión	74
12 Apendice	76
Diferencias contra BDE	76
Tipos de datos soportados	76
Internacioanlización y regionalización	78
Limitaciones	79
 Index	 81

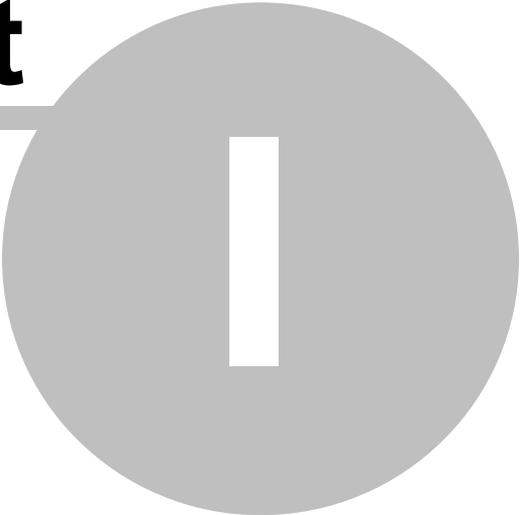
Foreword

This is just another title page
placed between table of contents
and topics

Top Level Intro

This page is printed before a new top-level chapter starts

Part



Tuesday, August 4, 2009 Accuracer

1.1 Introduction

Accuracer: Alto rendimiento unico archivo de base de datos multi usuario con SQL para Delphi, C++ Builder, Kylix y ODBC

Accuracer es el sistema mas rapido con la base de datos de unico archivo y cliente servidor con soporte para SQL

Caracteristicas principales:

- Base de datos de un solo archivo
- Acceso multi-usuario y Multi-Hilos
- Motor Cliente-Servidor
- No BDE; no DLLs;
- SQL'92 con subquerys e instrucciones DDL
- Soporta Integridad Referencial (Cumple con SQL'99)
- Ingenieria inversa (Exporta tablas a script SQL)
- Código fuente completo incluido
- Inigualables demos fáciles de usar
- De bajo impacto
- 100% compatibilidad con entandar controles DB-aware
- Sin regalías
- Soporte BLOB y Varchar (con compresión opcional)
- Soporta tablas en memoria (incluyendo SQL y DDL)
- Componente BatchMove
- Encriptación severa de el archivo de base de datos (AES, Blowfish, Twofish, DES, etc.)
- Backup y Restore (respaldo y restauración)
- Triggers - eventos en base de datos y servidor
- Manejador ODBC disponible
- Plataformas Windows/Linux: versiones disponibles en Delph, C++ Builder y Kylix
- Soporta transacciones, nivel de aislamiento READ COMMITTED

Visite nuestro sitio Web para leer las ultimas noticias y ver especificacion detallada detailed specification:

<http://www accuracer.com>

o

<http://www.aidaim.com>

1.2 Características

Funcionalidad

- Simple archivo de base de datos
- Soporte multi-usuario y multi-tarea
- Motor Cliente-Servidor
- Un sub conjunto de **SQL'92 incluyendo operadores DDL** es soportado por el componente TACRQuery. Con Accuracer puede crear SQL scripts para crear tablas, insertar, editar y eliminar registros, recuperar datos con el comando

SELECT . Lea en el libro de referencia SQL en esta guía para conocer más acerca del SQL implementado en Accuracer

- Ingeniería inversa (Exporte tablas a scripts SQL)
- Avanzado **motor de búsqueda**. Accuracer soporta el operador 'LIKE' con comodines '%' y '_' así como 'IS NULL' y 'IS NOT NULL' en filtros y queries
- Completo soporte para **múltiples índices**, es decir muchos campos de una tabla pueden conformar un índice. Accuracer provee índices ascendientes y descendientes, índices sensitivos y no sensitivos para campos string
- Tablas compartibles in-memory. Accuracer soporta acceso simultáneo a tablas por múltiples componentes TAccuracer desde una simple aplicación.
- Soporta valores mínimos, máximos y por defecto.
- Soporta campos requeridos (NOT NULL constraint).
- Soporta índice Primary y unique.
- Campos Autoincrement con cantidades de ajustes (valores mínimo y máximo, incremento ciclado) basado en secuencias.
- Soporta tablas en memoria con SQL & DDL
- Componentes BatchMove
- RepairDatabase. está implementado en TACRDatabase.
- Soporta base de datos ejecutable.
- Soporta para Backup y Restore.
- Triggers - Eventos de base de datos y servidor.
- Soporta transacciones, nivel de aislamiento READ COMMITTED.

Compacticidad

- Bajo impacto en el código compilado aproximadamente 740 Kb, sin manejadores externos requeridos (como el BDE)
- Poco consumo de memoria por Accuracer o el motor de la base de datos.
- Soporte opcional de compresión para los campos tipo Varchar y WideVarchar.
- Repida compresión para datos BLOB. Sus grandes campos de datos necesitarán menos memoria. Accuracer puede comprimir datos en vivo. Las rutinas de compresión usadas en Accuracer son mucho más rápidas que la mayoría de los compresores más populares como PKZip, WinRar, Arj.
- El método CompactDatabase en TACRDatabase permite compactar el archivo de la base de datos.
- Automática reducción de la base de datos en caso de borrado de datos al final de base de datos.

Alto rendimiento

- Búsquedas rápidas por los índices B-tree.. Hasta el momento Accuracer es uno de las bases de datos más rápidas existentes para Delphi y C++ Builder.
- Alta-velocidad de operaciones en memoria es alcanzado principalmente por el especialmente optimizado algoritmo del manejador de memoria y afinación.
- Operaciones rápidas con strings. Accuracer compara strings hasta 3 veces más

requerido que las rutinas estén escritas en Delphi. El alto rendimiento es alcanzado por el uso de librerías especiales escritas con Assembler y algoritmos avanzados de clasificación.

- Avanzado **optimizador SQL** ofrece hacer la ejecución de queries significativamente más rápido por elegir el mejor plan de ejecución.

Compatibilidad

- Accuracer soporta la mayoría de los tipos de campo TTable, incluyendo campos BLOB, Por otra parte permite crear campos strings y wide de **cualquier longitud fija o variable**.
- Accuracer es totalmente compatible con controles visuales **standard DB-aware** como QuickReport, DBGrid, DBNavigator, DBImage, DBMemo, DBRichEdit,, como también con componentes vendidos de terceros que soporten descendencia TDataSet - FastReport, DBFlyTreeView y otros.
- Campos Calculated y lookup pueden ser usadas de la misma manera que con TTable.
- La mayoría de las funciones de TTable son soportadas incluyéndolos métodos **Key** y **Range**.

Conveniencia

- La **reestructuración** de tablas es la forma más fácil de resguardar todos los datos existentes.
- Importar datos desde y hacia con cualquier conjunto de datos es soportado. Accuracer provee con una forma simple para importar y exportar tablas usando el método ImportTable y ExportTable.
- Soporta Internacionalización / Regionalización. Todo el texto de búsqueda y funciones de clasificación usan la configuración regional del sistema, también regionalizar su aplicación con Accuracer es una tarea muy simple.
- Soporte **Unicode**. Todas las operaciones de texto trabajan con codificación multi-byte usando ftWideString.
- Ayuda comprensiva. Accuracer viene con una documentación completa presentada en Accuracer Developer's Guide y Accuracer Reference.
- Cantidad de demos para diferentes IDE - Delphi, C++ Builder, Kylix y ODBC.

Seguridad

- Encriptamiento de la base de datos por los mejores cifradores simétricos (AES, Blowfish, Twofish, DES, etc.)
- Configuración directa de los parámetros de encriptación - modos de cifrado (CTS, OFB, CBC, CFB), vector inicial, clave binaria.
- Soporta password (RipeMD128 / RipeMD256 hash used)
- Todas las páginas dentro de la base de datos están encriptadas, incluyendo todos los datos internos como índices, mapas y directorios.
- Páginas nuevas son llenadas por defecto con datos random.
- Segura generación de números random sobre el algoritmo LFSR.
- Implementación de algoritmos de código abierto (DEC 1 library por Hagen)

Reddmann)

Plataformas soportadas por el producto

- VCL - Delphi 4,5,6,7, 2005, 2006, 2007 y C++ Builder 4,5,6, 2006
- CLX - Kylix 3 Delphi
- ODBC

1.3 Solicitando ayuda y soporte técnico

Soporte Técnico Libre

Si tiene algunas preguntas, comentarios o ideas para agregar nuevas posibilidades y/o cambios en las funciones de los productos, contactenos fácilmente a support@aidaim.com.

Consideramos cualquier idea y podemos tomar esto en cuenta mientras desarrollamos nuevas versiones de nuestros productos.

Si encuentra problemas, por favor, informenos acerca de lo siguiente:

- Nombre del producto y versión.
- Información sobre el compilador: Delphi or C++ Builder, Version, Edition, Service Pack
- Ambiente informático: su OS y Service Pack
- Descripción de su problema (tanta información como sea posible para enterarnos bien del problema).
- Adjunte una prueba del proyecto donde el problema pueda ser reproducido (esto nos ayuda a resolver tan pronto como sea posible)

Típicamente el equipo de soporte de AidAim responde los mensajes en 24 horas, pero dependiendo de la singularidad y dificultad de su pregunta esto puede tomar un poco más de tiempo.

1.4 How to Buy

Para colocar una orden y tomar información sobre los precios, visita nuestro sitio www.aidaim.com.

Sientase en la libertad de contactarnos a support@aidaim.com si tiene algunas preguntas técnicas.

Para preguntas relacionadas con ventas contacte al Departamento de Ventas a sales@aidaim.com.

Si es un distribuidor o revendedor establecido y quiere tener los productos AidAim en su portafolio, por favor no dude en contactarnos a sales@aidaim.com

1.5 Trabajando con tablas

1.5.1 Creando un archivo de base de datos

Puede crear un archivo de base de datos usando uno de estos metodos:

- 1) Ejecute el utility ACRManager escoja New Database de las opciones de el menu Database. Siga las instrucciones.
- 2) Use el metodo CreateDatabase de el componente TACRDatabase. Ver el demo CreateDatabase.

1.5.2 Configurando los componentes de tabla y base de datos

Los pasos siguientes son instrucciones generales para configurar el componente de tabla en tiempo de dise o. Puede haber pasos adicionales que usted necesita adaptar a las características de una tabla a los requisitos de su aplicacion. Si necesita tabla in-memory no deberia crear un componente de base de datos, solo establezca la propiedad InMemory a True.

Para crear un componente de base de datos,

1. Coloque un componente TACRDatabase desde la pesta a de Accuracer en la paleta de componentes hacia el data module o sobre la forma.
2. Establezca la propiedad filename hacia el directorio de el archivo de base de datos. Puede user OpenFileDialog si pulsa doble click sobre esta propiedad en inspector de objetos.

Para crear un componente tabla,

1. Si necesita una tabla in-memory establezca la propiedad InMemory a True, de otra manera establezca la propiedad DatabaseName para especificar cual componente TACRDatabase sera usado para conectar a la base de datos.
2. Coloque un componente TCARTable desde la pesta a Accuracer de la paleta de componentes en un data module o un form, y establezca la propiedad Name a un valor unico apropiado para su aplicacion.
3. Establezca la propiedad TableName con el nombre de la tabla en la base de datos.

Para acceder la data encapsulada por el componente tabla,

1. Coloque un componente data source desde la pesta a Data Access de la paleta de componentes en el data module o form, y establezca la propiedad DataSet con el nombre de la tabla o componente.
2. Coloque un control data-ware, como un TDGrid sobre un form, y establezca la propiedad del control DataSource con el nombre de el componente data source colocado en el paso anterior.
3. Cambie la propiedad Connected del componente TACRDatabase a True, si usa una tabla desde un archivo de base de datos. Salte este paso si usa una tabla in-memory.
4. Coloque a True la propiedad del componente table.

1.5.3 Creando tablas

Introducción

Creación de tablas esta acoplado al metodo CreateTable de el componente TAccuracer. Las propiedades usadas por el metodo CreateTable incluyen FieldDefs, IndexDefs, TableName y la propiedad Exists.

Especificando los campos para crear

La propiedad FieldDefs es usada para especificar cuales campos seran definidos en la nueva tabla. La propiedad FieldDefs es un arreglo de objetos TFieldDef, cada uno de los cuales contiene informacion acerca de el campo a crear. Puede agregar nuevos objetos TFieldDefs usando el metodo Add de la propiedad FieldDefs. El metodo Add acepta los siguientes parametros para la definicion del campo:

Nombre de campo (String)	El parametro nombre de campo indica el nombre dado al campo.
Tipo de dato (TFieldType)	Parametro tipo de dato indica el tipo de dato dado al campo Available TFieldType data types
Tamaño (Word)	Parametro que indica el tamaño del campo. Este deberia usarse solo para campos String solamente. Para todos los otros tipos de datos este parametro deberia establecerse a 0. Para el tipo String este parametro indica el tamaño del campo. Para el tipo WideString este parametro indica el tamaño wide string del campo en bytes.
Requerido (Boolean)	Parametro que indica si nuevo campo deberia ser requerido (not null) mientras agrega o modifica registros.

FieldDefs Avanzados

Accuracer provee algunas funciones avanzadas que no pueden ser especificadas por FieldDefs como minimos, maximos y valores por defecto, campos de autoincremento, compresion de campos blob y varchar. Use AdvFieldDefs en lugar de FieldDefs para especificar campos con parametros avanzados. Si usa AdvFieldDefs entonces deberia vaciar la lista FieldDefs llamando el metodo Clear de FieldDefs y viceversa.

Vea la guia de referencia para mayor informacion acerca de IAdvFieldDefs (clase TACRAAdvFieldDef).

Especificando los indices a Crear

La propiedad IndexDefs es usada para especificar cuales indices seran definidos en la nueva tabla. La propiedad IndexDefs es un arreglo de objetos TIndexDef, cada uno de ellos contiene informacion acerca de los indices a crear. Puede agregar un nuevo objeto TIndexDef usando el metodo Add de objeto TIndexDefs contenido en la propiedad IndexDefs. El metodo Add acepta los siguientes parametros para definicion de indices:

Nombre de indice (String)	Parametro nombre de indice contiene el nombre dado al indice.
Lista de campos (String)	Parametro lista de campos contiene la lista de los campos incluidos dentro del indice, Múltiples nombres de campos indicados en este parametro deben ir separados por punto y coma (;).
Opciones de indice (TIndexOptions)	Parametro opciones de indice provee informacion acerca del tipo de indice a crear (por favor revise la referencia de componentes proveido con Accuracer para mayor informacion de las opciones disponibles en TIndexOptions).

Estableciendo la informacion de una tabla

La propiedad TableName especifica el nombre de la tabla a crear.

Creando la Tabla

Cualquier tabla debe tener una clave primaria sobre campos de cualquier tipo. Puede crear estas claves con el indice por medio de la opcion [ixPrimary]. El paso final in al creacion de tablas es llamar al metodo CreateTable. Es tambien recomendado que revise la la propiedad Exists del componente TAccuracer primero para estar seguro que no intenta recerar una tabla ya existente. El siguiente ejemplo muestra como crear la tabla CUSTOMER incluida en la base de datos DBDemos.adb' con el demo de Delphi usando el metodo CreateTable:

```
begin
  with MyAccuracer do
    begin
      TableName:='customer';
      with FieldDefs do
        begin
          Clear;
          Add('CustNo',ftAutoInc,0,False);
          Add('Company',ftString,30,False);
          Add('Addr1',ftString,30,False);
          Add('Addr2',ftString,30,False);
          Add('City',ftString,15,False);
          Add('State',ftString,20,False);
          Add('Zip',ftString,10,False);
          Add('Country',ftString,20,False);
          Add('Phone',ftString,15,False);
          Add('FAX',ftString,15,False);
          Add('TaxRate',ftFloat,0,False);
          Add('Contact',ftString,20,False);
          Add('LastInvoiceDate',ftDateTime,0,False);
        end;
      with IndexDefs do
        begin
          Clear;
          Add('PrimaryKey','CustNo',[ixPrimary]);
          Add('ByCompany','Company',[ixCaseInsensitive]);
        end;
      if not Exists then
        CreateTable;
      end;
    end;
end;
```

1.5.4 Abriendo y cerrando tablas

Introduction

Despu s de configurar el componente Accuracer y crear una tabla, precisamente abra la tabla para ver y editar la data en un control DB-Aware tal como un TDBGrid. Hay dos maneras de abrir una tabla. Puede establecer a True la propiedad Active, o puede llamar el metodo Open. El siguiente ejemplo muestra como usar el metodo Open para abrir una tabla llamada 'customer' con el componente llamado TAccuracer.

```
begin
  // table settings
  with MyAccuracer do
    begin
      TableName:='customers';
```

```

    ReadOnly:=False;
    Open;
  end;
end;

```

La propiedad `ReadOnly` causa que la tabla actual con el nombre indicado en la propiedad `TableName` sea abierta read-only (solo-lectura), lo cual indica a la aplicación en curso que está inhabilitada para modificar el contenido de la tabla hasta que la tabla sea cerrada y re-abierta con acceso de escritura (`ReadOnly=False`).

Estas son dos maneras de cerrar la tabla. Puede establecer la propiedad `Active` a `False`, o puede llamar el método `Close`. Controles activos asociados con el data source de la tabla serán dejados en blanco.

```

begin
  MyAccuracer.Close;
end;

```

Nota Importante:

Tabla In-Memory no es borrada después de cerrar la tabla. Debe llamar `DeleteTable` para la remoción física de la misma.

1.5.5 Navegando las tablas

Introducción

Estos son los métodos básicos que puede usar en el código de su aplicación para moverse a diferentes registros:

Método	Descripción
First	Mueve a la primera línea de la tabla.
Last	Mueve a la última línea de la tabla.
Next	Mueve a la siguiente línea de la tabla.
Prior	Mueve a la línea previa de la tabla.
SetRecNo	Mueve el cursor al registro especificado (RecNo es calculado con todos los filtros aplicados a la tabla)

Nota

Todos los métodos están basados en el actual índice.

Además a estos métodos, la siguiente tabla describe dos propiedades Booleanas de la tabla que permiten obtener información cuando interactúa a través de los registros.

Las propiedades `BOF` y `EOF` indican cuando el apuntador de registros está al principio o el final de la tabla respectivamente.

El siguiente código ilustra una de las maneras de obligar al código a un ciclo de proceso de registros para un componente Accuracer llamado `CustTable`:

```

CustTable.First; Va al primer registro, el cual establece la propiedad EOF a False
while not CustTable.EOF do Ciclo hasta que EOF es igual a True
begin
  Procesa cada registro aquí
...

```

`CustTable.Next`; Siempre que halla un próximo registro EOF permanecerá `False`, hasta que no existan más registros y se coloca a `True`
end;

1.5.6 Filtrando registros

Introducción

Establecer filtros sobre tablas está acompañado de varios métodos de el componente `TAccuracer`. La propiedad básica de filtrado incluye `Filter`, `FilterOptions` y la propiedad `Filtered`. El evento `OnFilterRecord` es usado para implementar un evento de filtrado de retorno que puede ser usado para filtrar registros usando código en Delphi. Todas las operaciones de filtros mantienen el orden del índice activo

Configurando la propiedad `Filter`

Para crear un filtro usando la propiedad `Filter`, establezca el valor de la propiedad con un string que contenga la condiciones de filtrado. El string contiene la condición del filtro. Por ejemplo, la siguiente instrucción crea un filtro que prueba el campo `State` de una tabla y revisa si este contiene un valor de `state` como `California`.

```
table1.Filter := 'State = ' + QuotedStr('CA');
```

También puede proveer un valor para `Filter` basado en el texto introducido en un control. Por ejemplo, la siguiente instrucción asigna el texto de un edit box a `Filter`

```
table1.Filter := Edit1.Text;
```

También puede crear una condición con campos booleanos:

```
table1.Filter := 'Married = TRUE';
```

Puede crear un string basado en ambos código de texto y sobre datos entrados por el usuario en un control:

```
table1.Filter := 'State = ' + QuotedStr(Edit1.Text);
```

Puede comparar valores de campos a literales, y a constantes usando los siguientes operadores lógicos de comparación:

Operator	Meaning
<	Menor que
>	Mayor que
>=	Mayor e igual que
<=	Menor e igual que
=	Igual a
<>	Diferente
AND	Todos los elementos evaluados deben resultar <code>True</code>
NOT	Los elementos evaluados deben retornar cualquiera resultado menos el indicada
OR	Al menos uno de dos elementos evalúan a <code>True</code>
[NOT] LIKE	Operador extendido para comparar campos strings con comodines %
IS [NOT] NULL	Operador extendido para evaluar si un campo es o no null

Usando combinaciones de los operadores antes indicados, se pueden crear muy sofisticados filtros. Por ejemplo, la siguiente instrucción revisa si las dos condiciones coinciden cuando sea aplicada a un registro,

```
(Custno > 1400) AND (Custno < 1500);
```

Estableciendo opciones de filtros

La propiedad FilterOptions lo habilita para especificar si o no un filtro que compara campos basados en string acepta registros basados en comparaciones parciales y si o no comparaciones de string son sensitivos a mayusculas. FilterOptions es un conjunto de propiedades que puede estar en un estado vacio (por defecto), o que pueden contener cualquiera o ambos de los siguientes valores:

<u>Value</u>	<u>Meaning</u>
<i>foCaseInsensitive</i>	Ignora diferencias entre mayusculas y minusculas cuando compara strings
<i>foPartialCompare</i>	Deshabilita la comparacion parcial (e.g. no empareja strings que finalizan con un asterisco (*))

Por ejemplo, la siguiente instrucción establece un filtro que ignora la diferencia entre mayusculas y minusculas cuando compare los valores de el campo State:

```
FilterOptions := [foCaseInsensitive];
```

```
Filter := "State" = "CA";
```

Activando el filtro

Establezca la propiedad Filtered a True

Cuando el filtrado esta activado, solamente aquellos registros que reunan el criterio del filtro estar an disponibles en la aplicación. El filtrado esta siempre en una condicion temporal ya que puede apagar el filtrado estableciendo la propiedad Filtered a False en cualquier momento.

1.5.7 Buscando registros

Introducción

Buscar registros en una tabla esta acompañado de varios métodos de el componente TAccuracer. Los métodos básicos de búsqueda son FindFirst, FindLast, FindNext, y FindPrior.

Puede usar también los métodos Locate y Lookup para ubicar un registro buscado.

Todas las operaciones de búsquedas usan el actual índice en curso.

Búsquedas pueden también realizarse sobre tablas con relaciones master/detail y tablas filtradas.

Búsqueda por métodos de hallazgo-

Buscar registros con un componente Accuracer por métodos de hallazgo es un proceso de tres pasos:

1. Estableciendo la propiedad de filtrado
2. Asigne las opciones de filtros para campos basados en string, si es necesario.
3. Invoque algunos de los siguientes métodos navegacionales: FindFirst(), FindLast(), FindNext(), y FindPrior().

Todos estos métodos navegacionales colocan el indicador de registro a un registro que empareje (si hubiere), colocandolo en el actual, y retornando True. Si un emparejamiento no es encontrado, la posición del apuntador de registros no es movido (permanece tal como estaba), y este método retorna False. Puede chequear el estado de la propiedad *Found* para envolver estas llamadas, y solamente toma acción cuando *Found* es True. Por ejemplo, si el apuntador del registro ya está sobre el registro buscado en la tabla, y ud., llama FindNext, el método retornará False, y el registro actual permanece sin cambios.

Usando Locate

Locate mueve el cursor a la primera línea emparejada con un conjunto de criterio de búsqueda. En esta simple forma, pasa el nombre de una columna para buscar, el valor de campo a emparejar, y una opción especificando si la consulta es sensitiva a mayúsculas y minúsculas o si este puede usar emparejamientos parciales de claves. Por ejemplo, el siguiente código mueve el cursor a la primera línea en la tabla CustTable donde el valor en la columna Company es "Professional Drivers, Ltd.":

```
var
  LocateSuccess: Boolean;
  SearchOptions: TLocateOptions;
begin
  SearchOptions := [loPartialKey];
  LocateSuccess := CustTable.Locate('Company', 'Professional Drivers, Ltd.', SearchOptions);
end;
```

Si Locate consigue una pareja, el primer registro conteniendo la pareja se convierte en el registro actual. Locate retorna True si este empareja con algún registro, False si esto no ocurre. Si una búsqueda falla, el registro actual permanece sin cambios.

El poder real de Locate viene dentro del juego cuando quiere buscar sobre múltiples columnas y especifica múltiples valores para buscar. Los valores de búsqueda son variants, los cuales habilitan a especificar diferentes tipos de datos en su criterio de búsqueda. Para indicar múltiples columnas en un string de búsqueda, separe los items individuales con un punto y coma (;)

Porque valores de búsqueda son variants, Si pasa múltiples valores, debe pasar un arreglo de tipo variant como un argumento (por ejemplo, el valor de retorno desde un método Lookup), o puede construir el arreglo variant en el vuelo usando la función VarArrayOf. El siguiente código ilustra una búsqueda sobre múltiples columnas usando múltiples valores de búsqueda y un emparejamiento de tipo parcial:

```
with CustTable do
  Locate('Company;Contact;Phone', VarArrayOf(['Sight Diver','P']), loPartialKey);
```

Locate usa el método más rápido posible para emparejarse con un registro.

Using Lookup

Lookup searches for the first row that matches specified search criteria. If it finds a matching row, it forces the recalculation of any calculated fields and lookup fields associated with the dataset, then returns one or more fields from the matching row. Lookup does not move the cursor to the matching row; it only returns values from it.

In its simplest form, you pass Lookup the name of field to search, the field value to match, and the field or fields to return. For Ejemplo, the following code looks for the first record in the CustTable where the value of the Company field is "Professional Drivers, Ltd.", and returns the company name, a contact person, and a phone number for the company:

```

var
  LookupResults: Variant;
begin
with CustTable do
  LookupResults := Lookup('Company', 'Professional Divers, Ltd.', 'Company; Contact; Phone');
end;

```

Lookup returns values for the specified fields from the first matching record it finds. Values are returned as variants. If more than one return value is requested, Lookup returns a variant array. If there are no matching records, Lookup returns a Null variant. For more information about variant arrays, see the online help.

The real power of Lookup comes into play when you want to search on multiple columns and specify multiple values to search for. To specify strings containing multiple columns or result fields, separate individual fields in the string items with semi-colons.

Because search values are variants, if you pass multiple values, you must either pass a variant array type as an argument (for Ejemplo, the return values from the Lookup method), or you must construct the variant array on the fly using the VarArrayOf function. The following code illustrates a lookup search on multiple columns:

```

var
  LookupResults: Variant;
begin
with CustTable do
  LookupResults := Lookup('Company; City', VarArrayOf(['Sight Diver', 'Christiansted']),
'Company; Addr1; Addr2; State; Zip');
end;

```

Lookup also uses the fastest possible method to locate the matching record.

1.5.8 Clasificando registros

Introduction

You may use the IndexName and IndexFieldNames properties to set the current index order, and consequently, sort the current table based upon the index definition for the selected index order. The IndexName property is used to set the name of the current index. If this property is set to blank (") records are not sorted and shown in physical order. The following Ejemplo shows how you would set the current index order to an index called CustomerName:

Use las propiedades IndexName e IndexFieldNames para establecer el índice de clasificación actual, y por consecuencia, ordenar la tabla basada en la definición del índice seleccionado. La propiedad IndexName es usada para establecer el nombre del índice actual. Si esta propiedad está en blanco (") los registros no serán clasificados y serán mostrados en el orden físico de grabación. El siguiente ejemplo muestra cómo establecer el índice de clasificación actual llamado CustomerName:

```

begin
with MyAccuracer do
  begin
    IndexName:='CustomerName';
    do something
  end;
end;

```

Por favor note que cambiando el índice de clasificación puede causar que el actual apuntador de registro se mueva a una diferente posición en la tabla (pero no necesariamente se mueve fuera de el registro actual a menos que el registros halla sido borrado o modificado por otro usuario), también por favor asegurarse de invocar el método *First* después de establecer la propiedad `IndexName` si quiere tener el apuntador de registro colocado al comienzo de la tabla sobre el próximo índice de clasificación. Puesto que los números lógicos de registros estan basados sobre el índice de clasificación el número de registro puede también cambiar. Si intenta cambiar la propiedad `IndexName` a ningún índice una excepción puede ser disparada.

La propiedad `IndexFieldNames` es usada para establecer el índice de clasificación actual especificando los nombres de campos de el índice deseado en lugar del nombre de índice. Múltiples nombres de campo deberán ser separados con un punto y coma. Usando la propiedad `IndexFieldNames` es deseable en casos donde se esta tratando de establecer el índice de clasificación actual basado sobre un conjunto de campos desconocidos y no tener ningún conocimiento de los índices de clasificación disponibles. La propiedad `IndexFieldNames` intentara emparejar el número de campos dado con el mismo número de campos que comienzan en alguno de los índices primarios o secundarios. El siguiente ejemplo muestra como deberá establecer el actual índice de clasificación a un índice secundario llamado `CustomerName` que consiste de el campo `CustomerName` y el campo `CustomerNo`:

```
begin
  with MyAccuracer do
    begin
      IndexFieldNames:='CustomerName;CustomerNo';
      do something
    end;
  end;
```

Observe por favor que si Accuracer no puede encontrar cualquier índice que empareje los nombres de campo deseados una excepción será levantada. Si usted está utilizando este método de fijar el orden actual del índice usted debe también estar preparado para atrapar esta excepción y para ocuparse de ella apropiadamente.

1.5.9 Uso de campos BLOB

Introducción

La compresión de campos BLOB es transparente, también puede facilmente usar esto si almacena grandes cantidades de datos en campos BLOB.

Uso

Use `TACRBlobStream` para acceder o modificar el valor de un campo BLOB en una Accuracer. `TACRBlobStream` es un objeto stream que provee servicios permitiendo a la aplicación leer desde o escribir hacia un objeto de campo que representa un campo Binary Large Object (BLOB).

`TACRBlobStream` permite a los objetos no tener un conocimiento especializado de como la data es almacenada en un campo BLOB asi como leer y escribir data y el empleo de un mecanismo de stream uniforme.

Para usar un BLOB stream, cree una instancia de `TACRBlobStream`, use el método de el stream para leer o escribir los datos, y entonces cierre el BLOB stream. No haga uso de la misma instancia de `TACRBlobStream` para acceder data para mas de un registro. En su lugar cree un nuevo objeto `TACRBlobStream` cada vez que necesite leer o escribir datos BLOB sobre un nuevo registro.

Ejemplo

El siguiente ejemplo lee los datos desde un campo memo dentro de un stream blob y despliega este en un control memo.

```
procedure TForm1.Button2Click(Sender: TObject);  
var  
    Buffer: PChar;  
    MemSize: Integer;  
    Stream: TACRBlobStream;  
begin  
    Stream := TACRBlobStream.Create(MyAccuracer.FieldByName('Notes') as TBlobField,  
    bmRead);  
    try  
        MemSize := Stream.Size;  
        Inc(MemSize); Hace espacio para el buffer.  
        Buffer := AllocMem(MemSize); Asignación de memoria..  
        try  
            Stream.Read(Buffer^, MemSize); Lee el campo notas dentro del buffer..  
            Memo1.SetTextBuf(Buffer); Despliega el contenido del buffer.  
        finally  
            FreeMem(Buffer, MemSize);  
        end;  
    finally  
        Stream.Free;  
    end;  
end;
```

1.5.10 Campos Varchar y BLOB

Introducción

Accuracer soporta campos string de tamaño variable - tipo de campo SQL varchar o tipo de campo ftString. Campos varchar permiten hacer más compacto el archivo de la base de datos, ya que ellos almacenan solamente el número de caracteres reales por cada campo de cada registro, mientras los campos de caracteres fijos siempre almacenan el máximo número de caracteres. También los campos varchar así como los campos BLOB pueden ser opcionalmente comprimidos. Usaremos terminología SQL para calificar los tipos de campos y así evitar malos entendidos. Campos Char es un campo de cantidad fija de caracteres, mientras Varchar es campo de cantidad variable de caracteres.

La compresión de campos BLOB y Varchar es transparente, fácilmente puede usarla si almacena grandes cantidades de datos.

Uso

Puede especificar una configuración de compresión para campos BLOB y Varchar:

- 1) Usando el utility ACRManager
- 2) Usando la propiedad AdvFieldDefs de TACRTable - vea la Guía de referencia, clase TACRAdvFieldDef.
- 3) Usando script SQL para la creación de tablas - vea el típico instrucción CREATE TABLE

Como elegir una configuración de compresión

Primero debe elegir uno de los tres algoritmos de compresión. SLIB, BZIP o PPM. ZLIB es el más rápido, pero provee la más baja tasa de compresión. Sin embargo se recomienda cuando tiene que almacenar datos cortos (~1 kb o menos). BZIP es un poco más lento, pero usualmente provee mejor compresión que ZLIB. Ambos algoritmos descomprimen datos (operación de lectura) mucho más rápido que comprimir datos (operación de escritura). La diferencia de velocidad puede ser hasta de 10 veces.

PPM es un algoritmo mucho más lento, ofreciendo mejor tasa de compresión sobre mayor tipos de datos. PPM descomprime datos un poco más lento que durante la compresión. Este algoritmo es recomendado para grandes cantidades de datos (100kb o más)

Después debe elegir el apropiado modo de compresión - valor entero desde 1 (mínima tasa de compresión y máxima velocidad) hasta 9 (máxima tasa de compresión y menor velocidad)

El parámetro tamaño de bloque BLOB es usado solamente por campos BLOB (o Memo), pero no para campos Varchar. Esto requiere un tamaño de buffer de memoria para el streaming de compresión. Valor por defecto es 100kb. Si establece un tamaño menor de memoria el buffer será menor, pero esto puede conducir a una lenta tasa de compresión y velocidad. Grandes valores ofrecerán mejor tasa de compresión (especialmente para PPM).

Nota: El uso de la memoria está determinado por el algoritmo de compresión y el modo. Un modo elevado requiere gran cantidad de memoria en BZIP y PPM. PPM requiere desde 2 MB a 100 Mb de RAM para operar, BZIP requiere desde 100kb. hasta 900kb de RAM y ZLIB requiere hasta 256kb de RAM. Si usa campos BLOB(o memo). Accuracer asigna la memoria al buffer el cual es especificado en el parámetro BLOB block Size (100 kb por defecto). Campos Varchar no asignan memoria a los buffers.

Nota: Si no se especifica algoritmo de compresión (none) el valor del modo es ignorado, mientras el parámetro block size de los campos BLOB aun permanece activo.

Como elegir entre los tipos de campos Varchar, Char, Memo y BLOB

Si necesita almacenar valores binarios se debe usar un campo BLOB.

Para datos de texto esto generalmente depende de las operaciones que se necesitan hacer con estos datos y sobre su tamaño.

Si necesita algún tipo de búsqueda o filtrado sobre el contenido del texto desde usar siempre los tipos de campo Varchar o Char. Búsquedas en campos Memo no está soportada (esto podría hacerse usando el evento OnFilterRecord, pero esto operaría mucho más lento).

Si no necesita hacer búsquedas y sus datos no serán muy grandes entonces debe usar un campo Memo.

Campos Varchar trabajan más rápido que los campos Memo y usualmente reportan mejor tasa de compresión.

Campos Char proveen mejor rendimiento que los campos Varchar, pero requieren mayor espacio en disco, especialmente cuando tienen muchos campos vacíos o con valores muy cortos. Los campos Varchar requieren desde 6 a 30 bytes por valor para el enlace necesario de la información almacenada y encabezados. Valores Null toman una reserva cerca de 6 bytes para cada enlace con el registro, valores no null toman adicionalmente de 20-24 bytes para el encabezado dependiendo de el tamaño de la data comprimida y si es más pequeño del tamaño de página (24 bytes) o no (20 bytes más el tamaño del espacio vacío hasta la última página).

1.6 Operaciones Avanzadas con Tablas

1.6.1 Resestructurando una tabla

Introducción

Reestructuración de tablas es ejecutado por el método `RestructureTable` del componente `TACRTable`. Las propiedades usadas por el método `RestructureTable` incluyen las propiedades `RestructureFieldDefs` y `RestructureIndexDefs`.

Especificando la nueva estructura de campos

La propiedad `RestructureFieldDefs` es usada para especificar cuales campos definen la tabla reestructurada. La propiedad `RestructureFieldDefs` es un arreglo de objetos `TFieldDef`, cada uno de los cuales contiene información acerca de los campos. Cuando la tabla es abierta la propiedad `RestructureFieldDefs` contiene la definición de los todos los campos existentes. No debe definir todos los campos. Solamente debe agregar, modificar y eliminar definiciones de campos. Puede añadir nuevos objetos `TFieldDef` invoque el método `Add` de el objeto `TFieldDefs` almacenado en la propiedad `FieldDefs`. El método `Add` acepta los siguientes parámetros para la definición de campos:

Nombre de campo (String)	Parámetro que indica el nombre que será dado al campo.
Tipo de dato (TFieldType)	El parámetro tipo de dato indica el tipo del campo disponible en la propiedad <code>DataType</code> en <code>TFieldType</code> <code>ftInteger</code> , <code>ftSmallInt</code> , <code>ftFloat</code> , <code>ftDateTime</code> , <code>ftBLOB</code> , <code>ftString</code> (cualquier string de longitud fija)
Tamaño (Word)	Parámetro que indica el tamaño del campo. Solo se debe usar para los campos tipo string solamente. Para todos los demás tipos este parámetro debe conservarse con valor 0. Para los campos tipo string indica el tamaño en caracteres del campo.
Requerido (Boolean)	El parámetro requerido indica cuando un nuevo campo debe ser requerido (no Null) durante la modificación o inclusión de registros.

Specifying the Indexes in a Restructured Table

La propiedad `RestructureIndexDefs` es usada para especificar cual índice será definido para la tabla reestructurada. La propiedad `RestructureIndexDefs` es un arreglo de objetos `TIndexDefs`, cada uno de ellos contiene información relacionada al índice a crear. Cuando la tabla es abierta la propiedad `RestructureIndexDefs` contiene las definiciones de los índices de la tabla abierta. No debe crear todos los índices. Puede añadir, modificar o eliminar alguna definición solamente. Puede crear nuevos objetos `TIndexDef` usando el método `Add` de el objeto `TIndexDefs` contenido en la propiedad `IndexDefs`. El método `Add` acepta los siguientes parámetros para el índice que comienza a definirse:

Nombre de índice (String)	Parámetro nombre de índice contiene el nombre a ser dado al índice
Lista de campos (String)	El parámetro lista de campos contiene la lista de campos a ser incluida dentro del índice. Múltiples nombres de campos indicados en este parámetro deben ser separados por un punto y coma (;)
Opciones del índice (TIndexOptions)	El parámetro opciones de índice indica información acerca de el tipo de índice que comenzará a crearse (por favor vea el component reference suplida con Accuracer para mayor información sobre las opciones disponibles de <code>TIndexOption</code>)

Reestructurando tablas

El paso final en la reestructuración de una tabla es invocar el método `RestructureTable`. El siguiente ejemplo muestra como reestructurar la tabla `CUSTOMER.DAT` incluida con el demo de Delphi usando el método `RestructureTable`:

(La estructura de esta tabla esta descrita en el ejemplo del tipo creando tablas)

```
MyAccuracer.Open;
MyAccuracer.Close;
with MyAccuracer do
  begin
    Modifica los campos de la estructura
    with RestructureFieldDefs do
      begin
        // Agregar nuevo campo
        Add('Customer Name',aftString,300,False);
        // Estableciendo un nuevo tamaño para el campo 'Company'
        Find('Company').Size := 100;
        // Eliminando el campo 'Birthday'
        DeleteFieldDef('Birthday');
      end;
    modificando las definiciones de indice
    with RestructureIndexDefs do
      begin
        // agregando un nuevo indice para el nombre de campo Customer
        Add('CustomerName_Index','Customer Name',[ixCaseInsensitive]);
        // actualizando el indice primario
        Find('PrimaryKey').Fields := 'Customer ID';
      end;
    // cambia solamente los campos de la estructura, no modifica otros parámetros como la
    encriptación
    RestructureTable;
  end;
MyAccuracer.Open;
```

1.7 Referencia SQL

1.7.1 Descripción

Introducción

Accuracer soporta el subconjunto de comandos SQL'92. Incluyen la mayoría de las instrucciones SQL'92 para la manipulación y definición de datos - SELECT, INSERT, UPDATE, DELETE, CREATE TABLE, DROP TABLE, CREATE INDEX, DROP INDEX. Accuracer contiene el componente nativo TACRQuery que permite ejecutar comandos y scripts SQL en forma flexible y rápida. Accuracer no usa ningún componente o manejadores de terceros. Este aprovechamiento permite conseguir alto rendimiento sobre la mayoría de los queries SQL y hace la distribución de su aplicación mucho más flexible. Sin BDE, sin dlls, sin necesidad de manejadores.

Para acceder a tablas in-memory especifique la opción MEMORY antes de abrir la table en cualquier instrucción SQL.

Puede ejecutar Scripts con el componente TACRQuery - justo asigne script de texto a la propiedad SQL de TACRQuery.

Comandos SQL deben ser separados por punto y coma (;).

Comandos SQL Soportados:

- SELECT Statement
- INSERT Statement
- UPDATE Statement
- DELETE Statement
- CREATE DATABASE Statement
- DROP DATABASE Statement
- CREATE TABLE Statement
- ALTER TABLE Statement
- DROP TABLE Statement
- CREATE INDEX Statement
- DROP INDEX Statement
- START TRANSACTION Statement
- COMMIT Statement
- ROLLBACK Statement

Vea También:

Convención de nombres
Operadores
Funciones

1.7.2 Convención de nombres

Introducción

El SQL de Accuracer soporta la forma más flexible de nombrar las bases de datos, tablas y columnas.

Nombres de tablas

Nombre de tablas deben ser una simple palabra o múltiples palabras. Múltiples palabras deben ser separadas por una comillas sencillas o dobles, o corchetes []. Por ejemplo:

```
SELECT *  
FROM "Detail Parts"
```

Puede usar la correlación de nombres:

```
SELECT DP.PartNo  
FROM [Detail Parts] DP
```

Column Name:

El SQL de Accuracer soporta una palabra o múltiples palabras para el nombre de columna, Múltiples palabras deben ser encerradas por una comillas sencillas o dobles, o corchetes []. Nombres de columnas que dupliquen palabras reservadas SQL son también reservadas. Por ejemplo:

```
SELECT Orders.[Cust No]  
FROM Orders
```

Puede hacer una pequeña correlación para los nombres de columnas:

```
SELECT C.Name AS CustName
FROM Customer C
WHERE CustName LIKE 'Bill%'
```

Comentarios

Puede usar comentarios en el texto del querys SQL para mantener alguna información importante acerca del query.

Línea sencilla de comentario debe ser comenzada con los símbolos '--':

```
-- Estas es una línea sencilla de comentario
SELECT * FROM CUSTOMERS
```

Otra variante de los comentarios es encerrar el texto entre símbolos /* y */. Esto puede ser usado temporalmente removiendo algunas partes del query:

```
SELECT * FROM CUSTOMERS
/* WHERE (Name = 'Mike') */
ORDER BY CustNo
```

Palabras reservadas

Here is the list of words reserved by Accuracer's SQL Engine. Some of them are not really supported, but reserved for further implementations.

Esta es una lista de las palabras reservadas por el motor SQL de Accuracer. Algunos de estos no son realmente soportados, pero están reservadas para una futura implementación.

```
'ABSOLUTE'
,'ACTION'
,'ADD'
,'ALL'
,'ALLOCATE'
,'ALTER'
,'AND'
,'ANY'
,'ARE'
,'AS'
,'ASC'
,'ASSERTION'
,'AT'
,'AUTHORIZATION'
,'AVG'
,'BEGIN'
,'BETWEEN'
,'BIT'
,'BIT_LENGTH'
,'BOTH'
,'BY'
,'CASCADE'
,'CASCADED'
,'CASE'
,'CAST'
,'CATALOG'
,'CHAR'
,'CHARACTER'
,'CHAR_LENGTH'
```

, 'CHARACTER_LENGTH'
, 'CHECK'
, 'CLOSE'
, 'COALESCE'
, 'COLLATE'
, 'COLLATION'
, 'COLUMN'
, 'COMMIT'
, 'CONNECT'
, 'CONNECTION'
, 'CONSTRAINT'
, 'CONSTRAINTS'
, 'CONTINUE'
, 'CONVERT'
, 'CORRESPONDING'
, 'COUNT'
, 'CREATE'
, 'CROSS'
, 'CURRENT'
, 'CURRENT_DATE'
, 'CURRENT_TIME'
, 'CURRENT_TIMESTAMP'
, 'CURRENT_USER'
, 'CURSOR'
, 'DATE'
, 'DAY'
, 'DEALLOCATE'
, 'DEC'
, 'DECIMAL'
, 'DECLARE'
, 'DEFAULT'
, 'DEFERRABLE'
, 'DEFERRED'
, 'DELETE'
, 'DESC'
, 'DESCRIBE'
, 'DESCRIPTOR'
, 'DIAGNOSTICS'
, 'DISCONNECT'
, 'DISTINCT'
, 'DOMAIN'
, 'DOUBLE'
, 'DROP'
, 'ELSE'
, 'END'
, 'END-EXEC'
, 'ESCAPE'
, 'EXCEPT'
, 'EXCEPTION'
, 'EXEC'
, 'EXECUTE'
, 'EXISTS'
, 'EXTERNAL'
, 'EXTRACT'
, 'FALSE'
, 'FETCH'
, 'FIRST'

, 'FLOAT'
, 'FOR'
, 'FOREIGN'
, 'FOUND'
, 'FROM'
, 'FULL'
, 'GET'
, 'GLOBAL'
, 'GO'
, 'GOTO'
, 'GRANT'
, 'GROUP'
, 'HEX'
, 'HAVING'
, 'HOUR'
, 'IDENTITY'
, 'IF'
, 'IMMEDIATE'
, 'IN'
, 'INDICATOR'
, 'INITIALLY'
, 'INNER'
, 'INPUT'
, 'INSENSITIVE'
, 'INSERT'
, 'INT'
, 'INTEGER'
, 'INTERSECT'
, 'INTERVAL'
, 'INTO'
, 'IS'
, 'ISNULL'
, 'ISOLATION'
, 'JOIN'
, 'KEY'
, 'LANGUAGE'
, 'LAST'
, 'LEADING'
, 'LEFT'
, 'LEVEL'
, 'LIKE'
, 'LOCAL'
, 'LOWER'
, 'MATCH'
, 'MAX'
, 'MEMORY'
, 'MIME64'
, 'MIN'
, 'MINUS'
, 'MINUTE'
, 'MODULE'
, 'MONTH'
, 'NAMES'
, 'NATIONAL'
, 'NATURAL'
, 'NCHAR'
, 'NEXT'

, 'NO'
, 'NOFLUSH'
, 'NOT'
, 'NULL'
, 'NULLIF'
, 'NUMERIC'
, 'OCTET_LENGTH'
, 'OF'
, 'ON'
, 'ONLY'
, 'OPEN'
, 'OPTION'
, 'OR'
, 'ORDER'
, 'OUTER'
, 'OUTPUT'
, 'OVERLAPS'
, 'PAD'
, 'PARTIAL'
, 'POSITION'
, 'PRECISION'
, 'PREPARE'
, 'PRESERVE'
, 'PRIMARY'
, 'PRIOR'
, 'PRIVILEGES'
, 'PROCEDURE'
, 'PUBLIC'
, 'READ'
, 'REAL'
, 'REFERENCES'
, 'RELATIVE'
, 'RESTRICT'
, 'REVOKE'
, 'RIGHT'
, 'ROLLBACK'
, 'ROWS'
, 'SCHEMA'
, 'SCROLL'
, 'SECOND'
, 'SECTION'
, 'SELECT'
, 'SESSION'
, 'SESSION_USER'
, 'SET'
, 'SIZE'
, 'SMALLINT'
, 'SOME'
, 'SPACE'
, 'SQL'
, 'SQLCODE'
, 'SQLERROR'
, 'SQLSTATE'
, 'START'
, 'SUBSTRING'
, 'SUM'
, 'SYSTEM_USER'

```
, 'TABLE'  
, 'TEMPORARY'  
, 'THEN'  
, 'TIME'  
, 'TIMESTAMP'  
, 'TIMEZONE_HOUR'  
, 'TIMEZONE_MINUTE'  
, 'TO'  
, 'TOP'  
, 'TRAILING'  
, 'TRANSACTION'  
, 'TRANSLATE'  
, 'TRANSLATION'  
, 'TRIM'  
, 'TRUE'  
, 'UNION'  
, 'UNIQUE'  
, 'UNKNOWN'  
, 'UPDATE'  
, 'UPPER'  
, 'USAGE'  
, 'USER'  
, 'USING'  
, 'VALUE'  
, 'VALUES'  
, 'VARCHAR'  
, 'VARYING'  
, 'VIEW'  
, 'WHEN'  
, 'WHENEVER'  
, 'WHERE'  
, 'WITH'  
, 'WORK'  
, 'WRITE'  
, 'YEAR'  
, 'ZONE'  
, 'PASSWORD' // for DDL commands  
, 'BLOBBLOCKSIZE' // for DDL commands  
, 'BLOBCOMPRESSIONMODE' // for DDL commands  
, 'BLOBCOMPRESSIONALGORITHM' // for DDL commands  
, 'LASTAUTOINC' // for DDL commands  
, 'MODIFY' // alter table blablabla modify ...  
, 'NEW' // for NEW PASSWORD in ALTER TABLE  
, 'INDEX' // for CREATE INDEX ...  
, 'NOCASE' // for CREATE INDEX ... NOCASE ..  
, 'LTRIM'  
, 'RTRIM'  
, 'POS'  
, 'LENGTH'  
, 'SYSDATE' // DateTime function  
, 'NOW'  
, 'TOBLOB' // TOBLOB function  
, 'TODATE' // TODATE function  
, 'TOSTRING' // TOSTRING function  
, 'AUTOINDEXES' // for DDL AutoIndexes  
, 'NOAUTOINDEXES' // for DDL AutoIndexes  
, 'INCREMENT'
```

```
, 'LASTVALUE'  
, 'MAXVALUE'  
, 'MINVALUE'  
, 'CYCLED'  
, 'NOMAXVALUE'  
, 'NOMINVALUE'  
, 'NOCYCLED'  
, 'INITIALVALUE'  
, 'RENAME'  
, 'QUARTER'  
, 'WEEKDAY'  
, 'DAYOFWEEK'  
, 'DAYNAME'  
, 'MONTHNAME'  
, 'MSECOND'  
, 'ABS'  
, 'CEILING'  
, 'CEIL'  
, 'FLOOR'  
, 'MOD'  
, 'POWER'  
, 'POW'  
, 'RANDOM'  
, 'RAND'  
, 'ROUND'  
, 'SIGN'  
, 'TRUNCATE'  
, 'TRUNC'  
, 'SHL'  
, 'SHR'  
, 'DATABASE'  
, 'FILE'  
, 'PAGESIZE'  
, 'MAXSESSIONSCOUNT'  
, 'CUMSUM'  
, 'CUMPROD'  
, 'GROUP_CONCAT'
```

1.7.3 Usando parámetros

Parámetros pueden ser usados para reemplazar valores de datos en la instrucción SQL. Los parámetros están identificados por la precedencia de dos puntos (:). Puede usar parámetros con instrucciones SELECT, INSERT, UPDATE o DELETE. Algunos ejemplos de cómo puede usar parámetros:

```
With ACRQuery1 do  
begin  
  SQL.Clear;  
  SQL.Add('INSERT INTO customer_Sort (Company,Address,CustNo)');  
  SQL.Add('VALUES (:Company, :Address, :CustNo)');  
  Params[0].AsString := 'AidAim Software';  
  Params[1].AsString := 'US';  
  Params[2].AsInteger := 4;  
  ExecSQL;  
end;
```

```

With ACRQuery1 do
begin
  SQL.Clear;
  SQL.Add('SELECT * FROM orders WHERE PaymentMethod = :PayMethod ');
  ParamByName('PayMethod').AsString := 'Visa';
  Open;
end;

```

1.7.4 Operadores

Introducción

Accuracer SQL supports these operator categories:
SQL Accuracer soporta estas categorías de operadores:

- Operadores Aritméticos
- Operadores de comparación
- Operadores Lógicos
- Operadores de concatenación de String

Operadores Aritméticos

Operadores Aritméticos realizan operaciones matemáticas sobre dos expresiones de cualquier de los tipos de datos categorizados como numéricos

<u>Operador</u>	<u>Significado</u>
+ (Suma)	Adición
- (Resta)	Sustracción
* (Multiplicación)	Multiplicación
/(División)	División

Ejemplo:

```

SELECT (Price * Quantity) AS Total
FROM Order

```

Bitwise Operators

Bitwise operators perform bitwise operations on the integer data types.

Name	Syntax	Description
SHL	SHL or <<	Bitwise shift left; moves the bits to the left, it discards the far left bit and assigns 0 to the right most bit.
SHR	SHR or >>	Bitwise shift right; moves the bits to the right, discards the far right bit and, if unsigned, assigns 0 to the left most bit, otherwise sign extends.
XOR	XOR or ^	Bitwise exclusive OR; compares two bits and generates a 1 result if the bits are complementary, otherwise it returns 0.
MOD	%, MOD	The same as MOD function.
AND	&	Bitwise AND.
OR		Bitwise OR.
NOT	~	Bitwise NOT.

Operadores de comparación

Los operadores de comparación prueban si o no dos expresiones son la misma. Operadores de comparación pueden ser usados sobre todas las expresiones excepto expresiones de los tipos de datos **BLOB, Memo, fmyMemo o Gráficos**

<u>Operadores</u>	<u>Significado</u>
=, ==	Igual que
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
<>, !=	Diferente a

El resultado de una comparación de operadores tiene que ser del tipo de dato Booleano, el cual tiene tres valores TRUE, FALSE y UNKNOWN (desconocido).
 Expresiones que retornan un tipo de dato Booleano son conocidas como expresiones Booleanas. Si un operador que tiene uno o dos expresiones NULL retornar UNKNOWN
 Expresiones con tipos de dato Booleano son usadas en la cláusula WHERE para filtrar los registros que califican cada registro buscado.

Ejemplo:

```
SELECT *
FROM Orders
WHERE (TaxRate > 0)
```

Operadores Lógicos

Operadores Lógicos prueban la veracidad de alguna condición. Operadores Lógicos como los operadores de comparación, retorna un tipo de datos Booleano con un valor TRUE o FALSE.

<u>Operador</u>	<u>Significado</u>
AND, &&	TRUE si ambas expresiones booleanas son verdaderas.
BETWEEN	TRUE si el operando esta el un rango.
IN	TRUE si el operando aparece en una lista de valores.
LIKE	TRUE si el operando coincide con un patron.
NOT, !	Invierte el valor de cualquier otro operador
OR	TRUE si alguna expresión es evaluada a verdadero
IS NULL, = NULL	TRUE si la expresión booleana es desconocida (UNKNOWN).
IS NOT NULL, <> NULL	FALSE si la expresión booleana es desconocida (UNKNOWN).
= ""	TRUE if StringArgument IS NULL.
<> ""	FALSE if StringArgument IS NOT NULL.

Ejemplos::

```
SELECT *
```

```

FROM Customer
WHERE (Company LIKE '%Club%')

SELECT *
FROM Orders
WHERE (ShipToCity IS NOT NULL)

SELECT *
FROM Orders
WHERE (TaxRate BETWEEN 0 and 5) AND (AmountPaid > 1)

SELECT *
FROM Orders
WHERE (ShipVIA IN ('UPS', 'DHL'))

```

Operadores concatenadores de String

El operador concatenador de string permite concatenar caracteres con el signo de suma (+) o con el signo de concatenación (||), el cual es también conocido como el operador de concatenación.

Ejemplo:

```

SELECT (FirstName + ' ' + LastName) AS Name
FROM Customers

```

1.7.5 HEX constants

Both C++ and Pascal style of hex constants.

Syntax;

0xff, \$ff

Example:

```

drop table test1;
create table test1 (int1 SmallInt, int2 SmallInt);
insert into test1 values (0xff,$ff);
insert into test1 values (0x00,$0f);
insert into test1 values (0x01,$002);
insert into test1 values (0x03,$0002);
select int1, int2, int1 XOR int2 as int3, int2 ^ int1 as int4,
HEX(int1 XOR int2) as str1, HEX(int2 ^ int1,1) as str2, HEX(int2 ^
int1,2) as str3
from test1 order by int1;

```

1.7.6 Funciones

Las siguientes son los tipos de funciones SQL

- Funciones agregadas
- Funciones de fecha y hora
- Funciones Miscelaneas
- Funciones Mathematical

- Funciones String
- Funciones de Conversion de tipos

1.7.6.1 Funciones agregadas

Nombre	Sintaxis	Descripción
AVG	AVG ([DISTINCT] expression)	Retorna el promedio de los valores en un grupo. Valores nulos ser n ignorados.
COUNT	COUNT ([DISTINCT] expression *)	Retorna el número de items en un grupo.
GROUP_CONCAT	GROUP_CONCAT ([DISTINCT] [ASC] [DESC] expression [, Separator])	Returns the concatenated string field values for the group of records
MIN	MIN (expression)	Retorna el minimo valor en la expresi n.
MAX	MAX (expression)	Retorna el valor máximo en la expresi n.
SUM	SUM ([DISTINCT] expression)	Retorna la suma de todos los valores en la expresi on. SUM puede ser usado con columnas numericas solamente. Valores nulos seran ignorados.

1.7.6.1.1 Función AVG

Retorna el promedio de los valores en un grupo.

Sintaxis;

AVG ([DISTINCT] expression)

Argumentos:

expression

Es una expresi n de numeraci n exacta o de categoria n mérica aproximada. Funciones agregadas y subqueries no son permitidos.

Si la opci n DISTINCT es usada entonces SUM calculara solamente los valores diferentes de la *expression*

Ejemplos:

```
SELECT AVG(AmountPaid) FROM Orders WHERE PaymentMethod='Cash'
SELECT AVG(DISTINCT EmpNo) FROM Orders
```

1.7.6.1.2 Función COUNT

Retorna el numero de items en el grupo.

Sintaxis:

COUNT ([DISTINCT] expression | *)

Argumentos:

expression

Es una expresi n de cualquier tipo excepto **tipos BLOB**. Funciones agregadas y subqueries no estan permitidas.

Si la opción `DISTINCT` es usada entonces `SUM` calculará solamente los valores diferentes de la *expression*

*

Especifica que todas las líneas deben ser contadas para retornar el total de líneas en la tabla. `COUNT(*)` no tiene parámetros y no puede ser usada con `DISTINCT`. `COUNT(*)` retorna el número de líneas en una tabla específica sin eliminar duplicados. Esta cuenta separadamente cada línea, incluyendo líneas con valores nulos.

Ejemplos:

```
SELECT COUNT(*) FROM Orders
SELECT COUNT(OrderN0), ShipVIA FROM Orders GROUP BY ShipVIA
SELECT COUNT(DISTINCT Terms), ShipVIA FROM Orders GROUP BY ShipVIA
```

1.7.6.1.3 GROUP_CONCAT Function

Returns the concatenated string field values for the group of records.

Syntax:

`GROUP_CONCAT ([DISTINCT] [ASC] [DESC] expression [, Separator])`

Arguments:

expression

Is an expression based on the string field.

If `DISTINCT` options is specified then only different values will be concatenated.

`NULL` values are always skipped.

ASC or DESC

Specifies if the values should be sorted ascending or descending. `ASC` is the default setting.

Separator

Specifies the string value that will be used as a separator between values. Comma is the default separator.

You can use `GROUP_CONCAT` with `GROUP BY` or without it. In last case all records will be scanned as a single group.

Examples:

```
SELECT GROUP_CONCAT(name) FROM test
SELECT GROUP_CONCAT(DESC name) FROM test
SELECT num, GROUP_CONCAT(DISTINCT DESC name, "; ") FROM test GROUP BY num
```

1.7.6.1.4 Función MIN

Retorna el mínimo valor de la expresión.

Sintaxis:

`MIN (expression)`

Argumentos:

expression

Es una expresión de cualquier tipo excepto **tipos BLOB**. Funciones Agregadas y subqueries no están permitidas.

Ejemplos:

```
SELECT MIN(OrderNo) FROM Orders
SELECT MIN(Company) FROM Customer
```

1.7.6.1.5 Función MAX

Retorna el valor máximo de una expresión.

Sintaxis:

```
MAX ( expression )
```

Argumentos:

expression

Es una expresión de cualquier tipo excepto **tipos Blob**. Funciones agregadas y subqueries no son permitidas.

Ejemplo:

```
SELECT MAX(SaleDate) FROM Orders
```

1.7.6.1.6 Función SUM

Retorna la suma de todos los valores en la expresión. SUM debe ser usado con columnas numéricas solamente. Valores nulos serán ignorados.

Sintaxis:

```
SUM ( [DISTINCT] expression )
```

Argumentos:

expression

Es una expresión de categoría tipo de datos numérico exacto o numérico aproximado. Funciones agregadas y subqueries no son permitidas.

Si la opción DISTINCT es usada SUM calcula solamente valores diferentes de la *expression*.

Ejemplos:

```
SELECT SUM(AmountPaid) FROM Orders WHERE PaymentMethod='Visa'
SELECT SUM(DISTINCT EmpNo) FROM Orders
```

1.7.6.2 Funciones de Fecha y Hora

Nombre	Sintaxis	Descripción
CURRENT_DATE	CURRENT_DATE	Retorna la fecha actual de sistema.
CURRENT_TIME	CURRENT_TIME	Retorna la hora actual de sistema.
CURRENT_TIMESTAMP	CURRENT_TIMESTAMP	Retorna la fecha y hora actual de sistema.
MP	STAMP	
DAY	DAY (<i>expression</i>)	Retorna el día (entero de 1 - 31) extraída de la fecha en la expresión.
DAYNAME	DAYNAME (<i>expression</i>)	Retorna el nombre del día (valor string) extraído de la fecha en la expresión.
DAYOFWEEK	DAYOFWEEK (<i>expression</i>)	Retorna el día de la semana (valor entero 1 - Lunes, 2-Martes,.....,7 - Domingo) extraída de la fecha en la expresión.
EXTRACT	EXTRACT (<i>operator</i> FROM <i>expression</i>)	Extrae el año, trimestres, meses, días, horas, minutos, segundos, mili segundos de la fecha, hora y fecha hora en la expresión.

HOUR	HOUR (<i>expression</i>)	Retorna la hora (valor entero 0 - 23) extra da de la hora en la expresi n.
MINUTE	MINUTE (<i>expression</i>)	Retorna los minutos (valor entero; 0 - 59) extra dos de la hora en la expresi n.
MONTH	MONTH (<i>expression</i>)	Retorna el mes (valor entero: 1 - 12) extra do de la fecha en la expresi n.
MONTHNAME	MONTHNAME (<i>expression</i>)	Retorna el nombre de el mes (valor string) extra do de la fecha en al expresi n.
MSECOND	MSECOND (<i>expression</i>)	Retorna los mili segundos (valor entero: 0 - 999) extra dos de la hora en la expresi n.
NOW	NOW	Regresa la fecha y hora actual del sistema.
QUARTER	QUARTER (<i>expression</i>)	Retorna el trimestre del a o (valor entero: 1 - 4) extra do de la fecha en la expresi n.
SECOND	SECOND (<i>expression</i>)	Retorna los segundos (valor entero: 0 - 59) extra dos de la fecha en la expresi n.
SYSDATE	SYSDATE	Retorna la fecha y hora actual del sistema.
TODATE	TODATE(<i>StringValue,</i> <i>DateFormat</i>)	Convierte string a fecha usando en formato especificado.
TOSTRING	TOSTRING(<i>DateValue,</i> <i>DateFormat</i>)	Convierte fecha a string usando el formato especificado.
WEEKDAY	WEEKDAY (<i>expression</i>)	Retorna el d a de la semana (valor entero: 1 - Lunes, 2 - Martes,.....,7 - Domingo) extra do de la fecha en la expresi n.
YEAR	YEAR (<i>expression</i>)	Retorna el a o (valor entero) extra do de la fecha en la expresi n.

1.7.6.2.1 Funci3n CURRENT_DATE

Retorna la fecha actual de sistema.

Sintaxis:
CURRENT_DATE

Ejemplo:
SELECT LastInvoiceDate, CURRENT_DATE as CurDate
FROM Customer
WHERE LastInvoiceDate < NOW

1.7.6.2.2 Funci3n CURRENT_TIME

Retorna la hora actual de sistema.

Sintaxis:
CURRENT_TIME

Ejemplo:
SELECT LastInvoiceDate, CURRENT_TIME as CurTime
FROM Customer
WHERE LastInvoiceDate < NOW

1.7.6.2.3 Funciones CURRENT_TIMESTAMP, NOW AND SYSDATE

Retorna la fecha y hora actual de sistema.

Sintaxis:

```
CURRENT_TIMESTAMP  
NOW  
SYSDATE
```

Ejemplo:

```
SELECT LastInvoiceDate, NOW as CurDate  
FROM Customer  
WHERE LastInvoiceDate < NOW
```

1.7.6.2.4 Función DAY

Retorna el día (entero de 1 - 31) extraída de la fecha en la expresión.

Sintaxis:

```
DAY ( expression )
```

Argumentos:

expression

Es una expresión de tipo fecha, hora o fechahora que especifica la fecha fuente.

Ejemplo:

```
SELECT DAY(LastInvoiceDate) FROM Customer
```

1.7.6.2.5 Función DAYNAME

Retorna el nombre del día (valor string) extraído de la fecha en la expresión.

Sintaxis:

```
DAYNAME ( expression )
```

Argumentos:

expression

Es una expresión de tipo fecha, hora o fechahora que especifica la fecha fuente.

Ejemplo:

```
SELECT DAYNAME(LastInvoiceDate) FROM Customer
```

1.7.6.2.6 Función DAYOFWEEK

Retorna el día de la semana (valor entero 1 - Lunes, 2-Martes,.....,7 - Domingo) extraída de la fecha en la expresión.

Sintaxis:

```
DAYOFWEEK ( expression )
```

Argumentos:

expression

Es una expresi n de tipo fecha, hora o fechahora que especifica la fecha fuente.

Ejemplo:

```
SELECT DAYOFWEEK(LastInvoiceDate) FROM Customer
```

1.7.6.2.7 Funci3n EXTRACT

Extrae el a1o, trimestres, meses, d1as, horas, minutos, segundos, mili segundos de la fecha, hora y fechahora en la expresi3n.

Sintaxis:

EXTRACT (*operator* FROM *expression*)

EXTRACT (*operator*, *expression*)

Argumentos:

operator

Especifica el operador de extracci3n:

Operador	Descripci3n
DAY	Returns the day (integer value: 1 - 31) extracted from the date expression.
DAYNAME	Returns the name of the day (string value) extracted from the date expression.
DAYOFWEEK	Returns the day of week (integer value: 1 - Monday, 2 - Tuesday, ..., 7 - Sunday) extracted from the date expression.
HOUR	Returns the hours (integer value: 0 - 23) extracted from the time expression.
MINUTE	Returns the minutes (integer value: 0 - 59) extracted from the time expression.
MONTH	Returns the month (integer value: 1 - 12) extracted from the date expression.
MONTHNAME	Returns the name of the month (string value) extracted from the date expression.
MSECOND	Returns the milliseconds (integer value: 0 - 999) extracted from the time expression.
QUARTER	Returns the quarter of the year (integer value: 1 - 4) extracted from the date expression.
SECOND	Returns the seconds (integer value: 0 - 59) extracted from the time expression.
WEEKDAY	Returns the day of week (integer value: 1 - Sunday, 2 - Monday, ..., 7 - Saturday) extracted from the date expression.
YEAR	Returns the year (integer value) extracted from the date expression.

expression

Es una expresi n de tipo fecha, hora o fechahora que especifica la fecha fuente.

Ejemplos:

```
SELECT EXTRACT(YEAR FROM LastInvoiceDate) FROM Customer
```

```
SELECT EXTRACT(MONTH, LastInvoiceDate) FROM Customer
```

1.7.6.2.8 Funci3n HOUR

Retorna la hora (valor entero 0 - 23) extra da de la hora en la expresi3n.

Sintaxis:

HOUR (*expression*)

Argumentos:

expression

Es una expresi n de tipo fecha, hora o fechahora que especifica la fecha fuente.

Ejemplo:

```
SELECT HOUR(LastInvoiceDate) FROM Customer
```

1.7.6.2.9 Función MINUTE

Retorna los minutos (valor entero; 0 - 59) extraídos de la hora en la expresión.

Sintaxis:

MINUTE (*expression*)

Argumentos:

expression

Es una expresión de tipo fecha, hora o fechahora que especifica la fecha fuente.

Ejemplo:

```
SELECT MINUTE(LastInvoiceDate) FROM Customer
```

1.7.6.2.10 Función MONTH

Retorna el mes (valor entero: 1 - 12) extraído de la fecha en la expresión.

Sintaxis:

MONTH (*expression*)

Argumentos:

expression

Es una expresión de tipo fecha, hora o fechahora que especifica la fecha fuente.

Ejemplo:

```
SELECT MONTH(LastInvoiceDate) FROM Customer
```

1.7.6.2.11 Función MONTHNAME

Retorna el nombre de el mes (valor string) extraído de la fecha en la expresión.

Sintaxis:

MONTHNAME (*expression*)

Argumentos:

expression

Es una expresión de tipo fecha, hora o fechahora que especifica la fecha fuente.

Ejemplo:

```
SELECT MONTHNAME(LastInvoiceDate) FROM Customer
```

1.7.6.2.12 Función MSECOND

Retorna los mili segundos (valor entero: 0 - 999) extraídos de la hora en la expresión.

Sintaxis:

MSECOND (*expression*)

Argumentos:

expression

Es una expresi n de tipo fecha, hora o fechahora que especifica la fecha fuente.

Ejemplo:

```
SELECT MSECOND(LastInvoiceDate) FROM Customer
```

1.7.6.2.13 Funci3n QUARTER

Retorna el trimestre del a o (valor entero: 1 - 4) extra do de la fecha en la expresi n.

Sintaxis:

QUARTER (*expression*)

Argumentos:

expression

Es una expresi n de tipo fecha, hora o fechahora que especifica la fecha fuente.

Ejemplo:

```
SELECT QUARTER(LastInvoiceDate) FROM Customer
```

1.7.6.2.14 Funci3n SECOND

Retorna los segundos (valor entero: 0 - 59) extra dos de la fecha en la expresi n.

Sintaxis:

SECOND (*expression*)

Argumentos:

expression

Es una expresi n de tipo fecha, hora o fechahora que especifica la fecha fuente.

Ejemplo:

```
SELECT SECOND(LastInvoiceDate) FROM Customer
```

1.7.6.2.15 Funci3n TODATE

Convierte string a fecha usando en formato especificado

Sintaxis:

TODATE(*StringValue*, *DateFormat*)

Argumentos:

StringValue

Es una expresi n de tipo fecha, hora o fechahora que especifica la fecha fuente..

DateFormat

Es una expresi n de tipo string o wide string que especifica el formato de fecha para convertir el valor fecha a string.

String *DateFormat* son strings compuestos de indicadores que respresentan los valores a ser insertados dentro del string formateado. Algunos indicadores (como una "d"), simplemente formatean numero o strings. otros indicadores (como un "/") referido a el string local especifico.

En la siguiente tabla, los indicadores est n dados en may sculas. May sculas o min sculas son ignoradas en el formato.

<u>Indicador</u>	<u>Despliegue</u>
-	Despliega el separador de fecha '-'
/	Despliega el separador de fecha '/'
.	Despliega el separador de fecha '.'
,	Despliega el separador de fecha ','
:	Despliega el separador de fecha ':'
;	Despliega el separador de fecha ';'.
'TEXT'	Displays the text that will be included in the result of TOSTRING function without any conversion. The leading and trailing quotes will be omitted.
YYYY or YEAR	Despliega el año como un número de cuatro dígitos (0000-9999)
YY	Despliega el año como un número de dos dígitos (00-99).
Q	Despliega el trimestre de el año (1-4), 1 los meses Enero, Febrero, Marzo, 2 los meses Abril, Mayo y Junio, 3 los meses Julio, Agosto, Septiembre y 4 los meses Octubre, Noviembre y Diciembre.
MONTH	Despliega el nombre completo del mes (Enero-Diciembre),
MON	Despliega el nombre del mes abreviado (Ene-Dic).
MM	Despliega el mes como un número de dos dígitos (01-12).
M	Despliega el mes como un número sin ceros (1-12).
RM	Despliega el mes como un número romano (I - XII).
DDD	Despliega el día de el año sin ceros (1 - 366).
DD	Despliega el día del mes con un número de dos dígitos (01 - 31).
D	Despliega el día del mes sin ceros (1 - 31).
DAY	Despliega el día por su nombre (Lunes - Domingo).
DY	Despliega el día con una abreviación de su nombre (Sun - Dom).
DW	Despliega el día de la semana (1-7).
HH	Despliega la hora con un número con ceros (01 - 12).
HH12	
HH24	Despliega la hora con ceros de relleno (01 - 24).
H	Despliega la hora sin ceros de relleno (1 - 12).
H12	
H24	Despliega la hora sin ceros de relleno (1 - 24).
NN	Despliega el minuto con ceros (00:59).
N	Despliega el minuto sin ceros (0:59).
SS	Despliega el segundo con ceros (00:59).
S	Despliega el segundo sin ceros (0:59).
ZZZ	Despliega los mili segundos con ceros (000:999).
Z	Despliega los mili segundo sin ceros (0:999).
AMPM	Despliega el indicador de meridiano AM.

Ejemplo:

```
SELECT LastInvoiceDate, NOW as CurDate
FROM Customer
WHERE LastInvoiceDate < TODATE('12/16/2002 11:10:30 am', 'MM/DD/YYYY
hh:nn:ss ampm')
```

1.7.6.2.16 Función TOSTRING

Convierte fecha a string usando el formato especificado.

Sintaxis:

TOSTRING(*DateValue*, *DateFormat*)

Argumentos:*DateValue*

Es una expresión de tipo fecha, hora o fechahora que especifica la fecha fuente..

DateFormat

Es una expresión de tipo string o wide string que especifica el formato de fecha para convertir el valor fecha a string.

String *DateFormat* son strings compuestos de indicadores que representan los valores a ser insertados dentro del string formateado. Algunos indicadores (como una "d"), simplemente formatean número o strings. otros indicadores (como un "/") referido a el string local específico.

En la siguiente tabla, los indicadores están dados en mayúsculas. Mayúsculas o minúsculas son ignoradas en el formato.

<u>Indicador</u>	<u>Despliegue</u>
-	Despliega el separador de fecha '-'
/	Despliega el separador de fecha '/'.
.	Despliega el separador de fecha '.'.
,	Despliega el separador de fecha ','.
:	Despliega el separador de fecha ':'.
;	Despliega el separador de fecha ';'.
'TEXT'	Displays the text that will be included in the result of TOSTRING function without any conversion. The leading and trailing quotes will be omitted.
YYYY or YEAR	Despliega el año como un número de cuatro dígitos (0000-9999)
YY	Despliega el año como un número de dos dígitos (00-99).
Q	Despliega el trimestre de el año (1-4), 1 los meses Enero, Febrero, Marzo, 2 los meses Abril, Mayo y Junio, 3 los meses Julio, Agosto, Septiembre y 4 los meses Octubre, Noviembre y Diciembre.
MONTH	Despliega el nombre completo del mes (Enero-Diciembre),
MON	Despliega el nombre del mes abreviado (Ene-Dic).
MM	Despliega el mes como un número de dos dígitos (01-12).
M	Despliega el mes como un número sin ceros (1-12).
RM	Despliega el mes como un número romano (I - XII).
DDD	Despliega el día de el año sin ceros (1 - 366).
DD	Despliega el día del mes con un número de dos dígitos (01 - 31).
D	Despliega el día del mes sin ceros (1 - 31).
DAY	Despliega el día por su nombre (Lunes - Domingo).
DY	Despliega el día con una abreviación de su nombre (Sun - Dom).
DW	Despliega el día de la semana (1-7).
HH	Despliega la hora con un número con ceros (01 - 12).
HH12	
HH24	Despliega la hora con ceros de relleno (01 - 24).
H	Despliega la hora sin ceros de relleno (1 - 12).
H12	

H24	Despliega la hora sin ceros de relleno (1 - 24).
NN	Despliega el minuto con ceros (00:59).
N	Despliega el minuto sin ceros (0:59).
SS	Despliega el segundo con ceros (00:59).
S	Despliega el segundo sin ceros (0:59).
ZZZ	Despliega los mili segundos con ceros (000:999).
Z	Despliega los mili segundo sin ceros (0:999).
AMPM	Despliega el indicador de meridiano AM.

Ejemplo:

```
SELECT TOSTRING>LastInvoiceDate,'Today is' mm/dd/yyyy hh24:nn:ss:zzz '
Wow !!!') Formated_Date, LastInvoiceDate
FROM Customer
```

1.7.6.2.17 Función WEEKDAY

Retorna el día de la semana (valor entero: 1 - Lunes, 2 - Martes,.....,7 - Domingo) extraído de la fecha en la expresión.

Sintaxis:

```
WEEKDAY ( expression )
```

Argumentos:

expression

Es una expresión de tipo fecha, hora o fechahora que especifica la fecha fuente.

Ejemplo:

```
SELECT WEEKDAY>LastInvoiceDate) FROM Customer
```

1.7.6.2.18 Función YEAR

Retorna el año (valor entero) extraído de la fecha en la expresión.

Sintaxis:

```
YEAR ( expression )
```

Argumentos:

expression

Es una expresión de tipo fecha, hora o fechahora que especifica la fecha fuente.

Ejemplo:

```
SELECT YEAR>LastInvoiceDate) FROM Customer
```

1.7.6.3 Funciones Miscelaneas

Nombre	Sintaxis	Descripción
ISNULL	ISNULL (<i>expression</i> [, <i>replacement</i>])	Retorna el valor de reemplazo en caso de que el valor de la expresión sea NULL. Si el reemplazo no es especificado entonces retorna verdadero si la expresión es NULL y retorna falso si la expresión no es NULL.
LASTAUTOINC	LASTAUTOINC(<i>table_name</i> ,	Retorna el ultimo valor de autoincremento de la tabla indicada.

column_name)

1.7.6.3.1 Función ISNULL

Retorna el valor de reemplazo si la expresi n es NULL. Si un valor de reemplazo no es especificada retorna true si la expresi n es NULL y retorna false si expresi n no es NULL.

Sintaxis:

ISNULL (*expression* [, *replacement*])

Argumentos:

expression

Es cualquier expresi n valida.

replacement

Valor para reemplazar valores NULL. El tipo de el valor de reemplazo debe ser del mismo tipo de la expresion. Usela funcion CAST para convertir entre tipos de datos.

Ejemplos:

```
SELECT ISNULL(Addr2, 'No Address') FROM Customer
SELECT * FROM Customer WHERE ISNULL(Addr2)
SELECT SUM(ISNULL(AmountPaid, CAST(10.0, CURRENCY))) FROM Orders
```

1.7.6.3.2 Función LASTAUTOINC

La funci n LASTAUTOINC retorna el ultimo valor de auto incremento para la tabla indicada.

Sintaxis:

LASTAUTOINC(*table_name*, *column_name*)

Argumentos:

table_name

Es una constante string que indica el nombre de la tabla que tomara el valor de auto incremento.

column_name

Es una constante string indica el nombre del campo que tomara el ultimo valor de auto incremento dentro de la tabla.

Ejemplo:

```
INSERT INTO Employee (Name, DeptID)
VALUES ('John Smith', LASTAUTOINC( Department, ID ))
```

1.7.6.4 Mathematical Functions

Name	Syntax	Description
ABS	ABS (x)	Returns the absolute value of x.
SIGN	SIGN (x)	SIGN(x) returns the sign of input x as -1, 0, or 1 (negative, zero, or positive respectively).
MOD	MOD (x, y)	Returns the integer remainder of x divided by y (same as x%y).
FLOOR	FLOOR (x)	Returns the largest integer value that is less than or equal to x.
CEILING	CEILING (x) or CEIL (x)	Returns the smallest integer value that is greater than or equal to x.

CUMPRO	CUMPROD(x D)	Returns the cumulative product of all prior values and current value.
CUMSUM	CUMSUM (x)	Returns the cumulative sum of all prior values and current value.
ROUND	ROUND (x) or ROUND (x, d)	Returns the value of x rounded to the nearest whole integer. Returns the value of x rounded to the number of decimal places specified by the value d.
TRUNCA TE	TRUNCATE (x, d) or TRUNC (x, d)	Returns the number X, truncated to D decimals. If D is 0, the result will have no decimal point or fractional part.
POWER	POWER (x, y) or POW (x, y)	Returns the value of x raised to the power of y.
HEX	HEX(X [, MODE])	If X is a number, returns a string representation of the hexadecimal value of X, where X is integer. If X is a string, returns a hexadecimal string of X where each character in X is converted to 2 hexadecimal digits. MODE: 0 - just convert to hex (default), 1 - Pascal hex style (\$FF), 2 - C++ hex style (0xff).
RAND, RANDOM	RAND () or RAND (n)	Returns a random floating-point value in the range 0 to 1.0. If an integer argument N is specified, it is used as maximum value and result will be integer 0 <= X < N

1.7.6.4.1 ABS Function

Returns the absolute value of x.

Syntax;
ABS (x)

Arguments:

x
Is an expression of the numeric data type.

Examples:

```
SELECT abs(int1) as int_val, abs(float1) as float_Val from test1
```

1.7.6.4.2 SIGN Function

Returns the sign of input x as -1, 0, or 1 (negative, zero, or positive respectively).

Syntax;
SIGN (x)

Arguments:

x
Is an expression of the numeric data type.

Examples:

```
SELECT sign(int1) as int_val, sign(float1) as float_Val from test1
```

1.7.6.4.3 MOD Function

Returns the integer remainder of x divided by y (the same as x%y).

Syntax;
MOD (x)

Arguments:

x

Is an expression of the numeric data type.

Examples:

```
SELECT int1 % int2 as int_Val2, int1 MOD int2 as int_Val3 from test1
```

1.7.6.4.4 FLOOR Function

Returns the largest integer value that is less than or equal to x.

Syntax;
FLOOR (x)

Arguments:

x

Is an expression of the numeric data type.

Examples:

```
SELECT FLOOR(int1) as int_val, FLOOR(float1) as float_Val from test1
```

1.7.6.4.5 CEILING Function

Returns the absolute value of x.

Syntax;
CEILING (x)

Arguments:

x

Is an expression of the numeric data type.

Examples:

```
SELECT CEIL(int1) as int_val, CEILING(float1) as float_Val from test1
```

1.7.6.4.6 CUMSUM Function

Returns the cumulative sum of all prior values and current value.

Syntax;
CUMSUM (x)

Arguments:

x
Is an expression based on numeric field values.

Examples:

```
SELECT Cost, CUMSUM(Cost) FROM Orders
```

1.7.6.4.7 CUMPROD Function

Returns the cumulative product of all prior values and current value.

Syntax;

```
CUMPROD( x )
```

Arguments:

x
Is an expression based on numeric field values.

Examples:

```
SELECT Cost, CUMPROD(Cost) FROM Orders
```

1.7.6.4.8 ROUND Function

Returns the value of *x* rounded to the nearest whole integer or to the number of decimal places specified by the value *d*.

Syntax;

```
ROUND ( x ) or  
ROUND ( x, d )
```

Arguments:

x
Is an expression of the numeric data type.
d
Is an expression of the integer data type.

Examples:

```
SELECT float1, ROUND(float1) as float_Val1, ROUND(float1,1) as  
float_Val2, ROUND(float1,2) as float_Val3 from test1
```

1.7.6.4.9 TRUNCATE Function

Returns the number *X*, truncated to *D* decimals. If *D* is 0, the result will have no decimal point or fractional part.

Syntax;

```
TRUNCATE ( x )
```

Arguments:

x
Is an expression of the numeric data type.
d

Is an expression of the integer data type.

Examples:

```
SELECT float1, TRUNCATE(float1) as float_val1, TRUNC(float1,1) as  
float_val2, TRUNC(float1,2) as float_val3 from test1
```

1.7.6.4.10 POWER Function

Returns the value of x raised to the power of y .

Syntax;

```
POWER ( x, y ) or  
POW ( x, y )
```

Arguments:

x, y
Are an expressions of the numeric data type.

Examples:

```
SELECT POWER(int1,int2) as exp1, POW(float1,int2) as exp2 from test1
```

1.7.6.4.11 HEX Function

If X is a number, returns a string representation of the hexadecimal value of X , where X is integer.
If X is a string, returns a hexadecimal string of X where each character in X is converted to 2 hexadecimal digits.

Syntax;

```
HEX ( X [, MODE] )
```

Arguments:

x
Is an expression of the numeric data type.

MODE

0 - just convert to hex (default),
1 - Pascal hex style (\$FF),
2 - C++ hex style (0xff).

Examples:

```
select HEX(int1 XOR int2) as str1, HEX(int2 ^ int1,1) as str2, HEX(int2  
^ int1,2) from test1
```

1.7.6.4.12 RANDOM Function

Returns a random floating-point value in the range 0 to 1.0. If an integer argument n is specified, it is used as maximum value and result will be integer X , where $0 \leq X < n$.

Syntax;

RAND () or
 RAND (*n*) or
 RANDOM () or
 RANDOM (*n*)

Arguments:

n
 Is an expression of the integer data type.

Examples:

```
select test1.*, RAND(100000) as rnd from memory test1 order by num desc,id
select test1.*, RANDOM as rnd from memory test1 order by num desc,id
```

1.7.6.5 Funciones String

Nombre	Sintaxis	Descripción
LENGTH	LENGTH (<i>expression</i>)	Retorna el número de caracteres en un string excluyendo el terminador null.
LOWER	LOWER (<i>expression</i>)	Retorna una expresión de caracteres después de convertir los caracteres en minúsculas a mayúsculas.
LTRIM	LTRIM (<i>expression</i>)	Retorna una expresión de caracteres después de remover los espacios en blanco a la izquierda.
POS	POS (<i>substring</i> , <i>expression</i>)	Retorna el valor índice de el primer carácter indicado en un substring que ocurre en el string dado. Pos es sensible a las mayúsculas.
RTRIM	RTRIM (<i>expression</i>)	Retorna un string de caracteres después de truncar todos los espacios a la derecha.
SUBSTRING	SUBSTRING (<i>expression</i> , <i>startindex</i> [, <i>length</i>])	Retorna un substring de un string.
TRIM	TRIM (<i>expression</i>)	Retorna un string de caracteres después de truncar todos los espacios en blanco.
UPPER	UPPER (<i>expression</i>)	Retorna una expresión de caracteres con caracteres en minúsculas convertidos a mayúsculas.

1.7.6.5.1 Función LENGTH

Retorna el número de caracteres en un string excluyendo el terminador null.

Sintaxis:

LENGTH (*expression*)

Argumentos:

expression
 Es una expresión de tipo string o wide string.

Ejemplo:

```
SELECT * FROM Customer
WHERE LENGTH(Company) > 5
```

1.7.6.5.2 Función LOWER

Retorna una expresión de caracteres después de convertir los caracteres en minúsculas a mayúsculas.

Sintaxis:

LOWER (*expression*)

Argumentos:

expression

Es una expresión de tipo string o wide string.

Ejemplo:

```
SELECT LOWER(Company) FROM Customer
```

1.7.6.5.3 Función LTRIM

Retorna una expresión de caracteres después de remover los espacios en blanco a la izquierda.

Sintaxis:

LTRIM (*expression*)

Argumentos:

expression

Es una expresión de tipo string o wide string.

Ejemplo:

```
SELECT LTRIM(Company) FROM Customer
```

1.7.6.5.4 Función POS

Retorna el valor índice de el primer carácter indicado en un substring que ocurre en el string dado. Pos es sensible a las mayúsculas.

Sintaxis:

POS (*substring*, *expression*)

Argumentos:

substring

Es una expresión de tipo string o wide string que indica el substring de búsqueda en el indicado string.

expression

Es una expresión de tipo string o wide string que indica el string de origen.

Ejemplo:

```
SELECT * FROM Customer  
WHERE Pos('Blue',Company) > 0
```

1.7.6.5.5 Función RTRIM

Retorna un string de caracteres despues de truncar todos los espacios a la derecha.

Sintaxis:

RTRIM (*expression*)

Argumentos:

expression

Es una expresi n de tipo string o wide string.

Ejemplo:

```
SELECT RTRIM(Company) FROM Customer
```

1.7.6.5.6 Función SUBSTRING

Retorna un substring de un string.

Sintaxis:

SUBSTRING (*expression*, *startindex* [, *length*])

Argumentos:

expression

Is a an expression of string or wide string type.

startindex

Is a constant that specifies the character position at which the extracted substring starts within the original string.

length

Is a constant that specifies number of characters being extracted from source string.

Ejemplo:

```
SELECT SUBSTRING(Company, 2, 5)  
FROM Customer
```

1.7.6.5.7 Función TRIM

Retorna un string de caracteres despu s de truncar todos los espacios en blanco.

Sintaxis:

TRIM (*expression*)

Argumentos:

expression

Es una expresi n de tipo string o wide string.

Ejemplo:

```
SELECT TRIM(Company) FROM Customer
```

1.7.6.5.8 Función UPPER

Retorna una expresi n de caracteres con caracteres en min sculas convertidos a may sculas.

Sintaxis:

UPPER (*expression*)

Argumentos:

expression

Es una expresi n de tipo string o wide string.

Ejemplo:

```
SELECT UPPER(Company) FROM Customer
```

1.7.6.6 Funciones de conversi3n de tipos

Nombre	Sintaxis	Descripci3n
CAST	CAST(<i>value</i> , <i>data_type</i>)	La funci3n CAST convierte un indicado valor e el tipo de datos especificado.
TOBLOB	TOBLOB(<i>value</i> [, <i>format</i>])	La funci3n TOBLOB convierte un valor string a un valor BLOB.

1.7.6.6.1 Funci3n CAST

La funci3n CAST convierte un valor indicado a el indicado tipo de datos.

Sintaxis:

CAST(*value*, *data_type*)

Argumentos:

value

Es una expresi3n de cualquier tipo valido.

data_type

Es una constante que indica el tipo de dato para la conversi3n del valor especificado por el valor de la funci3n CAST puede ser usada con los siguientes tipos de datos:

<u>Tipo de dato</u>	<u>Descripci3n</u>
AutoInc	Auto incremento 32-bit entero sin signo.
BCD	N mero de coma flotante.
Currency	N mero de coma flotante.
Date	Valor fecha.
DateTime	Valor fechahora.
Float	N mero de coma flotante.
Integer	32-bit entero sin signo.
LargeInt	64-bit entero sin signo..
Logical	Valor booleano.
SmallInt	16-bit entero sin signo.
String	String de longitud fija (puede ser hasta 2^32 s mbolos).
Time	Valor hora.
WideString	String Unicode de longitud fija (puede ser hasta 2^32 s mbolos).
Word	16-bit entero sin signo.

1.7.6.6.2 Función TOBLOB

La función TOBLOB convierte el valor string indicado a un valor BLOB.

Sintaxis:

TOBLOB(*value* [, *format*])

Argumentos:

value

Es un valor string a ser convertido a un valor BLOB usando el formato indicado.

format

Dos formatos son soportados:

MIME64 -MIME64 formato estándar (usando en e-mails)

HEX - número de mayúsculas hexadecimal.

El formato por defecto es MIME64 (típicamente provee menor tamaño de string)

Ejemplo:

```
INSERT INTO jpeg VALUES (
    'AidAim',
    TOBLOB ('QWlkQWltIFNvZnR3YXJlDQpIZXJlIHRvIEh1bHANCg==',MIME64),
    NULL, 1);
```

1.7.7 Declaración SELECT

Introduction

La declaración SELECT es usada para recuperar datos desde las tablas.

Sintaxis

```
SELECT [DISTINCT | ALL] [TOP n [, first_row_number]]
* | column [AS correlation_name | correlation_name], [column...]
[INTO destination_table]
FROM table_reference [AS correlation_name | correlation_name] [PASSWORD
'password_string']
[[[NATURAL] [INNER | [LEFT | RIGHT | FULL] OUTER JOIN] table_reference [AS
correlation_name | correlation_name]
[ON join_condition] | USING (join columns)]
[WHERE predicates]
[GROUP BY group_fields_list]
[HAVING predicates]
[ORDER BY order_list]
[UNION [ALL] [CORRESPONDING [BY (column_list)]] SELECT...]
[EXCEPT | MINUS [ALL] [CORRESPONDING [BY (column_list)]] SELECT...]
[INTERSECT [ALL] [CORRESPONDING [BY (column_list)]] SELECT...]
```

La declaración SELECT indica la lista de las columnas a recuperar. Use un asterisco para recuperar todas las columnas.

The ALL option (Default) retrieves all rows from specified tables.

La opción ALL (defecto) recupera todas las líneas de la tabla indicada.

La opción DISTINCT recupera solo las líneas diferentes. Este es un simple ejemplo de como recuperar los diferentes contactos de la tabla customer.

```
SELECT DISTINCT Contact FROM Customer
```

The TOP option specifies that only the first n rows are to be output from the query result set. If first_row_number specified then n rows starting from this row number will be retrieved.

La opción TOP indica que solamente las primeras de n líneas son recuperadas como resultado del query. Entonces las líneas a partir del n número indicado son recuperadas.

La cláusula INTO indica un nombre de tabla donde la data recuperada por la declaración SELECT será almacenada.

La cláusula FROM indica la(s) tabla(s) de donde serán recuperada(s) las columna(s).

La cláusula WHERE indica las condiciones de filtrado para la declaración SELECT.

La cláusula GROUP BY divide el resultado en grupos.

```
SELECT CustNo, SUM(ItemsTotal)
FROM Orders
GROUP BY CustNo
```

La cláusula HAVING indicada una condición de búsqueda para un grupo o un agregado.

Cláusula ORDER BY

Sintaxis:

1) ORDER BY <order_list>

or

2) ORDER BY INDEX IndexName

<order_list> ::= [TableName.]FieldName [ASC | DESC] [NOCASE]

La cláusula ORDER BY indica el orden de las líneas recuperadas por el query.

Sintaxis 1 indica los campos específicos para ordenar el resultado.

Si las opciones no son indicadas el orden de clasificación será ascendente y sensitivo a mayúsculas.

Opción ASC indica clasificación ascendente para el este campo.

Opción DESC indica clasificación descendente para este campo.

Opción NOCASE indica clasificación sin reconocimiento de mayúsculas y minúsculas.

El siguiente ejemplo muestra como guardar líneas de la tabla Customers hacia una tabla nueva.

```
SELECT CustNo, Contact, Company, City, Country
INTO NewCustomer
FROM Customer
ORDER BY Country DESC, City DESC NOCASE, Company NOCASE, Contact
```

Sintaxis 2 permite especificar el nombre de un índice existente.

Esta sintaxis puede ser usada solamente para queries de una sola tabla.

Aquí está un Ejemplo:

```
SELECT * from Customer_findKey ORDER BY INDEX ByCompany
```

Clausulas Join, Natural y el uso de operadores

Estos son tres tipos de joins que puede ser usados en la cláusula FROM para hacer joins relacionales: Cartesian, Inner and Outer.

Un Cartesian join combina tablas fuentes sin ninguna correlación. El resultado de este join es una tabla conteniendo todas las columnas desde la tabla fuente y todas las combinaciones de todas las

líneas desde la tabla fuente. Por ejemplo, si la primera tabla contiene 5 registros y la segunda tabla contiene 10 registros el resultado de Cartesian join de estas tablas contendrá 50 registros, la sintaxis es como sigue:

```
FROM table_reference, table_reference [,table_reference...]
```

Aquí un ejemplo:

```
SELECT * FROM Members,Departments
```

Un Inner join incluye todas las líneas combinadas de la tabla fuente que tiene valores comunes de una especificada columna.

Un Inner join incluye todas las líneas combinadas desde la tabla fuente que tiene valores comunes de una especificada columna y líneas desde left, right o ambas tablas fuentes que no tiene línea correspondiente en otra tabla fuente. Los Outer join pueden ser LEFT, RIGHT o FULL.

Un Inner Join incluye todas las líneas combinadas desde la tabla fuente que tienen valores comunes de la columna indicada.

Un Outer join incluye todas las líneas combinadas desde la tabla fuente que tengan valores comunes con las columnas indicadas y líneas desde las tablas fuentes left, right o both que no tienen líneas correspondientes en otras tablas fuentes. Los Outer join pueden ser LEFT, RIGHT o FULL.

Las líneas sin correspondencia desde la tabla derecha contienen valor NULL para las columnas de la izquierda y viceversa.

Una de las principales ventajas de Accuracer es que soporta todos los tipos de JOINS: Cartesian, Inner, Left Outer, Right Outer, Full Outer.

Por otra parte, Accuracer provee alto rendimiento sobre joins debido a la flexible y bien diseñada arquitectura que excluye toda la transferencia de datos innecesario.

Estas son tres formas de indicar un inner o outer joins:

1) Las columnas comunes son indicadas en la cláusula ON:

```
FROM table_reference [INNER | LEFT | RIGHT | FULL] JOIN table_reference
ON predicate
[[INNER | LEFT | RIGHT | FULL] JOIN table_reference ON predicate...]
```

2) Las columnas comunes son indicadas usando un operador (todas las tablas fuentes debería tener estas columnas)

```
FROM table_reference [INNER | LEFT | RIGHT | FULL] JOIN table_reference USING
(column_name[,column_name...])
[[INNER | LEFT | RIGHT | FULL] JOIN table_reference USING
(column_name[,column_name...])...]
```

3) Las columnas comunes son todas las columnas desde la tabla fuente que que tienen los mismos nombres:

```
FROM table_reference NATURAL [INNER | LEFT | RIGHT | FULL] JOIN table_reference
[NATURAL [INNER | LEFT | RIGHT | FULL] JOIN table_reference...]
```

Aquí algunos ejemplos de joins:

```
SELECT Contact, Customer.CustNo, Company, Orders.OrderNo, Orders.CustNo
FROM Customer INNER JOIN Orders
ON (Customer.CustNo = Orders.CustNo)
WHERE Contact LIKE 'E%'
ORDER BY Contact,Orders.CustNo,Orders.OrderNo
```

```
SELECT Contact, Customer.CustNo, Company, Orders.OrderNo, Orders.CustNo
```

```
FROM Customer INNER JOIN Orders Using (CustNo)
ORDER BY Contact,Orders.CustNo,Orders.OrderNo
```

```
SELECT cb.*
FROM Customer_Base cb NATURAL INNER JOIN Customer_Base
```

```
SELECT Contact, Customer.CustNo, Company, Orders.OrderNo, Orders.CustNo
FROM Customer NATURAL LEFT JOIN Orders
WHERE State IS NOT NULL
ORDER BY Contact,Orders.CustNo,Orders.OrderNo
```

```
SELECT Contact, Customer.CustNo, Company, Orders.OrderNo, Orders.CustNo
FROM Customer NATURAL RIGHT JOIN Orders
WHERE State IS NOT NULL
ORDER BY Contact,Orders.CustNo,Orders.OrderNo
```

```
SELECT Contact, Customer.CustNo, Company, Orders.OrderNo, Orders.CustNo
FROM Customer NATURAL FULL JOIN Orders
WHERE State IS NOT NULL
ORDER BY Contact,Orders.CustNo,Orders.OrderNo
```

cl usula UNION

Combina el resultado de dos o mas querys dentro de un simple resultado compuesto de todas las l íneas al pertenecer a todos los querys en la union. Esta es una forma diferente de usar joins que combinan columnas entre dos tablas. La sintaxis es:

```
[UNION [ALL] [CORRESPONDING [BY (column_list)]] SELECT...]
```

Tres reglas b ásicicas para combinar el resultado de dos querys con UNION son:

- si la opci n CORRESPONDING no es indicada entonces el n mero y el orden de las columnas deben ser ide ticas en ambos querys combinados.
- si la opci n CORRESPONDING es indicada entonces las columnas mostradas deben existir en ambos querys.
- Los tipos de datos deben ser id nticos.

la opci n ALL incorpora todas las l íneas dentro de el resultado, incluyendo duplicados. Si no es indicada, las l íneas duplicadas son removidas.

Ejemplo:

```
SELECT Company FROM customer_Base
UNION
SELECT Company FROM customer_Filter
UNION
SELECT Company FROM customer_Range
```

CI usula EXCEPT (MINUS)

Retorna un conjunto de l íneas de resultado pertenecientes al primer query, excluyendo las l íneas id nticas con alguna en el segundo query, opcionalmente retiene los duplicados.

```
[EXCEPT [ALL] [CORRESPONDING [BY (column_list)]] SELECT...]
```

Tres reglas b ásicicas para EXCEPT son:

- si la opción CORRESPONDING no es indicada entonces el número y orden de las columnas deben ser idénticos en ambos queries.
- si la opción CORRESPONDING es indicada entonces las columnas especificadas deben existir en ambos queries.
- Los tipos de datos deben ser idénticos.

ALL option incorporates all rows into the results, including duplicates. If not specified, duplicate rows are removed

La opción ALL incorpora todas las líneas dentro del resultado, incluyendo duplicados. Si no es indicada, las líneas duplicadas serán removidas..

Ejemplo:

```
SELECT * FROM customer_Range
EXCEPT CORRESPONDING BY (Company)
SELECT * FROM customer_Filter
```

Cláusula INTERSECT

Retorna un conjunto de líneas pertenecientes a el primer query que tengan una pareja en el segundo query, opcionalmente retiene los duplicados.

```
[INTERSECT [ALL] [CORRESPONDING [BY (column_list)]] SELECT...]
```

Tres reglas básicas para INTERSECT son:

- si la opción CORRESPONDING no es utilizada entonces el número y orden de las columnas deben ser idénticos en ambos queries.
- si la opción CORRESPONDING es utilizada entonces las columnas indicadas deben existir en ambos queries.
- los tipos de datos deben ser idénticos.

La opción ALL incorpora todas las líneas dentro de el resultado, incluyendo duplicados. Si no es indicado, las líneas duplicadas serán removidas,

Ejemplo:

```
SELECT * FROM customer_Range
INTERSECT CORRESPONDING BY (Company)
SELECT * FROM customer_Filter
```

Cláusula WHERE

Indica las condiciones que deben ser satisfechas por todos las líneas recuperadas por el query. La cláusula WHERE puede incluir algunas funciones soportadas y operadores exceptuando las funciones agregadas.

Accuracer soporta solamente **subqueries no correlacionados**. i.e. no puede usar campos de un query pariente en un subquery.

Ejemplo de queries correlacionados - **no soportados por Accuracer**:

```
SELECT field1 FROM table1 T1
WHERE T1.field2 = (SELECT MAX(field1) FROM table2 T2 WHERE T2.field2 = T1.field3);
```

Vea unos Ejemplos en Utils\Bin\SQLConsole\SQL\SubQuery folder.

Ejemplo:

```
SELECT * from Jpeg
WHERE ID = (SELECT MIN(ID) from jpeg)
```

```
SELECT * FROM orders
WHERE CustNo IN
(SELECT DISTINCT CustNo FROM customer WHERE (Company LIKE 'S%') and (CustNo <
2500))
ORDER BY CustNo
```

```
SELECT Count(*) as ROW_COUNT FROM jpeg
WHERE EXISTS
(SELECT * FROM jpeg WHERE (Name LIKE 'A%'))
```

1.7.8 Declaración INSERT

Introducción

La declaración INSERT es usada para agregar uno o más líneas de datos en una tabla.

Sintaxis

```
INSERT INTO table_reference [password 'pass'] [(columns_list)]
VALUES (update_values)
```

Use la instrucción INSERT para agregar líneas de datos a una tabla específica.

La opción MEMORY indica que la tabla en memoria será creada.

La cláusula INTO indica la tabla que recibirá los datos insertados. columns_list, es una lista de columnas de la tabla y deben ir separadas por comas y encerradas en paréntesis y esto es opcional. Si la lista de columnas no es colocada los datos serán insertados dentro de todas las columnas de la tabla.

La cláusula VALUES indica la data que será insertada en la tabla.

Aquí está un ejemplo de cómo agregar registros usando la instrucción INSERT :

```
INSERT INTO Customer (CustNo,Company, City, State, Contact, LastInvoiceDate)
VALUES (5555,'AidAim Software','Phoenix','AZ','Ella Perelman','10/15/2002')
```

Para agregar líneas a una tabla que son recuperados desde otra tabla, omita la palabra clave VALUES y use un subquery como la fuente de datos de las nuevas líneas:

```
INSERT INTO Customer_Sort
SELECT * FROM Customer_Share
```

1.7.9 Declaración UPDATE

Introducción

La declaración UPDATE es usada para modificar uno o más líneas existentes en una tabla.

Sintaxis

```
UPDATE table_reference [Password "password_value"]  
SET column_ref = update_value [,column_ref = update_value...]  
[WHERE condition]
```

Use la instrucción UPDATE para modificar uno o mas valores de columnas de líneas existentes en una tabla especifica.

Use una referencia en la instrucción UPDATE para indicar la tabla que recibir los datos modificados.

La opción MEMORY indica que la tabla sera creada en-memoria.

La cláusula SET es una lista de expresiones separada por comas, para la instrucción UPDATE, La sintaxis es como sigue:

```
SET column_ref = update_value [,column_ref = update_value...]
```

Use la cláusula SET para indicar aquellas columnas que modificaran sus datos con los nuevos valores indicados.

La cláusula WHERE indica la condicion de filtrado para la instrucción UPDATE. La sintaxis es como sigue:

```
WHERE condition
```

Use la cláusula WHERE para actualizar solamente registros que cumplan con la condicion indicada.

Aquí esta un ejemplo:

```
UPDATE Members SET FirstName = 'New Name' WHERE ID >= 3
```

1.7.10 Declaración DELETE

Introduction

La declaración DELETE es usada para eliminar una o mas líneas de datos de una tabla especifica.

Sintaxis

```
DELETE  
FROM table_reference [PASSWORD 'password_string']  
[WHERE predicates]
```

La opción MEMORY indica que una tabla en-memoria sera creada.

La cláusula FROM indica la tabla a ser usada por la instrucción DELETE. La sintaxis como sigue:

```
FROM table_reference
```

La cláusula WHERE indica las condiciones de filtrado para la instrucción UPDATE. La sintaxis como sigue:

```
WHERE predicates
```

Use una cláusula WHERE para eliminar solamente aquellos registros que cumplan con la condición indicada.

Aquí está un ejemplo

```
DELETE FROM Members WHERE ID >= 3
```

1.7.11 CREATE DATABASE Statement

Introduction

The CREATE DATABASE statement is used to create a new database.

Syntax

```
CREATE DATABASE
    FILE "file_name" [ PAGESIZE {128..65535} ] [ MAXSESSIONSCOUNT {1..2147483648} ]
    |
    MEMORY database_name
```

Use CREATE DATABASE to create database with specified parameters.

You must specify FILE or MEMORY option to create disk or in-memory database, otherwise exception will be raised.

For disk database, file_name parameter is required. File name must be quoted by single quote (') or double quote (").

If PAGESIZE or MAXSESSIONSCOUNT is missed, default value will be used.

In case of in-memory database, you must specify the name of database only which can be quoted, double-quoted, or non-quoted.

Example 1.

```
CREATE DATABASE FILE "c:\temp\test.adb" PAGESIZE 8192 MAXSESSIONSCOUNT 10
- creates disk database with the parameters specified.
```

Example 2.

```
CREATE DATABASE MEMORY MemDB1
- creates in-memory database with the name MemDB1.
```

1.7.12 DROP DATABASE Statement

Introduction

The DROP DATABASE statement is used to delete the database.

Syntax

```
DROP DATABASE
    FILE "file_name" | MEMORY database_name
```

Use DROP DATABASE to delete database and all its data.

You must specify FILE or MEMORY option to delete disk or in-memory database, otherwise exception will be raised.

For disk database, file_name parameter is required. File name must be quoted by single quote (') or double quote (").

In case of in-memory database, you must specify the name of the existing database which can be quoted, double-quoted, or non-quoted.

Example 1.

```
DROP DATABASE FILE "c:\temp\test.adb"
- deletes database and its file c:\temp\test.adb.
```

Example 2.

```
DROP DATABASE MEMORY MemDB1
- deletes in-memory database with the name MemDB1.
```

1.7.13 Declaración CREATE TABLE

Introduction

La declaración CREATE TABLE es usada para crear nuevas tablas.

Sintaxis

```
CREATE TABLE table_name (
  column_name data_type [(dimensions)] |
    AutoInc([(data_type ]
      [ , INCREMENT integer ]
      [ , INITIALVALUE integer ]
      [ , MAXVALUE integer | NOMAXVALUE ]
      [ , MINVALUE integer | NOMINVALUE ]
      [ , CYCLED | NOCYCLED ]
    ])
  [ BLOBBLOCKSIZE {1..4294967295} ]
  [ BLOBCOMPRESSIONALGORITHM {NONE | ZLIB | BZIP | PPM} ]
  [ BLOBCOMPRESSIONMODE {0 .. 9} ]

  [ DEFAULT {const | NULL} ]
  [ NOT NULL | NULL ]
  [ PRIMARY [KEY] | UNIQUE [ ASC | DESC ] [ CASE | NOCASE ] ]
  [MINVALUE value | NOMINVALUE ]
  [MAXVALUE value | NOMAXVALUE ]
  [,column_name ...]

  [ , PRIMARY KEY [key name] (column_name [ ASC | DESC ] [ CASE | NOCASE ]
    [ { , column_name [ ASC | DESC ] [ CASE | NOCASE ] } ... ] ) ]
  [ , FOREIGN KEY [key name] (column_name [ { ,column_name } ... ] )
    REFERENCES table_name [MATCH FULL | MATCH PARTIAL]
    [ON DELETE <CASCADE | SET NULL | SET DEFAULT | NO ACTION>]
    [ON UPDATE <CASCADE | SET NULL | SET DEFAULT | NO ACTION>]]
)
```

table_name = [MEMORY] name_of_the_table.

Use CREATE TABLE para crear una tabla con la estructura indicada. table_name es el nombre de la tabla a ser creada. Si la opción MEMORY es indicada entonces una tabla en memoria será creada.

Column_name es el nombre de las columnas. Los tipos de datos deben ser uno de estos:

Valor	Descripción	Correspondiente TFieldType
Char, FixedChar	Fixed character field	ftFixedChar
Varchar, Varchar2, String	Character or variable length string field	ftString
WideChar, FixedWideChar	Fixed wide character field	ftWideString
WideVarchar, WideString	Wide character or variable length wide string field	ftWideString
Shortint, SignedInt8	8-bit integer field	ftSmallint
Smallint, SignedInt16	16-bit integer field	ftSmallint
Integer, SignedInt32	32-bit integer field	ftInteger
Largeint, Int64, SignedInt64	64-bit integer field	ftLargeint
Byte, UnsignedInt8	Byte field	ftWord
Word, UnsignedInt16	16-bit unsigned integer field	ftWord
Cardinal, UnsignedInt32	32-bit unsigned integer field	ftLargeint
AutoInc, AutoIncInteger	Auto-incrementing 32-bit integer counter field	ftAutoinc
AutoIncShortint	Auto-incrementing 8-bit integer counter field	ftAutoinc
AutoIncSmallint	Auto-incrementing 16-bit integer counter field	ftAutoinc
AutoIncLargeint	Auto-incrementing 64-bit integer counter field	ftAutoinc
AutoIncByte	Auto-incrementing byte counter field	ftAutoinc
AutoIncWord	Auto-incrementing 16-bit unsigned integer counter field	ftAutoinc
AutoIncCardinal	Auto-incrementing 32-bit unsigned integer counter field	ftAutoinc
Single	Single floating-point numeric field	ftFloat
Float, Double	Double floating-point numeric field	ftFloat
Extended	Extended floating-point numeric field	ftFloat
Boolean, Logical, Bool, Bit	Boolean field	ftBoolean
Currency, Money	Money field	ftCurrency
Date	Date field	ftDate
Time	Time field	ftTime
DateTime	Date and time field	ftDateTime
TimeStamp	Date and time field accessed through dbExpress	ftTimeStamp
Bytes	Fixed number of bytes (binary storage)	ftBytes
VarBytes	Variable number of bytes (binary storage)	ftVarBytes
Blob	Binary Large Object field	ftBlob
Graphic	Bitmap field	ftGraphic

Memo, Clob	Text memo field	ftMemo
FormattedMemo, FmtMemo	Formatted text memo field	ftFmtMemo
WideMemo, WideClob	Unicode text memo field	ftMemo

Las dimensiones es una tamaño del valor de la columna en bytes, Use este con tipos de datos bytes, string o wide string.

Use la opción NOT NULL para indicar columnas que pueden estar vacías.

Indique el nivel de compresión para el almacenamiento de campos BLOB (BLOB, FmtMemo, Memo, Graphic) a través de las opciones BlobCompressionAlgorithm y BlobCompressionMode

El BlobBlockSize es el tamaño en bytes del bloque de datos BLOB el cual es usado por el motor de la base de datos en operaciones de lecturas / escrituras con campos BLOB, Valor mínimo 1 byte valor por defecto 100kb.

Aquí está un ejemplo:

```
CREATE TABLE Test
(
  ID AutoInc PRIMARY KEY,
  Text String(500),
  Numeric Float,
  Money Currency,
  CurrentDate Date,
  Picture Graphic BlobCompressionAlgorithm ZLIB BlobCompressionMode 1
);
```

Nota:

Si el nombre de la tabla indicada a ser creada existe. CREATE TABLE despliega una excepción. Nombres de tabla, columna e índices pueden ser colocados entre corchetes ([]). Así puede usar palabras reservadas (como TABLE) y símbolos especiales (como ' ') en los nombres de tabla, columnas e índices.

1.7.14 Declaración ALTER TABLE

Introduction

La declaración ALTER TABLE es usada para modificar la estructura de una tabla existente.

Sintaxis

```
ALTER TABLE table_name ADD [COLUMN]
(
  column_name data_type [(dimensions)] [NOT NULL]
  [,column_name data_type [(dimensions)] [NOT NULL]...]
  [,PRIMARY KEY (column_name [, column_name...])]
)
ALTER TABLE table_name ADD
(
  [ PRIMARY KEY [key name] (column_name [ ASC | DESC ] [ CASE | NOCASE ]
    [, column_name...]) ]
  [ FOREIGN KEY [key name] (column_name [ { ,column_name } ... ] )
```

```

REFERENCES table_name [MATCH FULL | MATCH PARTIAL]
[ON DELETE <CASCADE | SET NULL | SET DEFAULT | NO ACTION>]
[ON UPDATE <CASCADE | SET NULL | SET DEFAULT | NO ACTION>]]
)

ALTER TABLE TableName <MODIFY> | <ALTER [COLUMN]> (
  column_name data_type [(dimensions)] |
    AutoInc[(data_type
      [, INCREMENT integer ]
      [, INITIALVALUE integer ]
      [, MAXVALUE integer | NOMAXVALUE ]
      [, MINVALUE integer | NOMINVALUE ]
      [, CYCLED | NOCYCLED ]
    )]
  [ BLOBBLOCKSIZE {1..4294967295} ]
  [ BLOBCOMPRESSIONALGORITHM {NONE | ZLIB | BZIP | PPM} ]
  [ BLOBCOMPRESSIONMODE {0 .. 9} ]

  [ DEFAULT {const | NULL} | DROP DEFAULT ]
  [ NOT NULL | NULL ]
  [ PRIMARY [KEY] | UNIQUE [ ASC | DESC ] [ CASE | NOCASE ]]
  [MINVALUE value | NOMINVALUE ]
  [MAXVALUE value | NOMAXVALUE ]
)

ALTER TABLE table_name DROP [COLUMN]
(
  column_name [,column_name...]
)

ALTER TABLE table_name DROP CONSTRAINT constraint_name [CASCADE | RESTRICT]

ALTER TABLE TableName RENAME [COLUMN] OldName [ TO ] NewName

ALTER TABLE TableName RENAME TO NewName
or
RENAME TABLE TableName TO NewName

```

Use ALTER TABLE para modificar la estructura de una tabla existente.
 La cl usula ADD es usada para agregar nuevas columnas a la tabla.
 Las cl usulas ALTER o MODIFY son usadas para modificar las definiciones de las columnas.

Nota: Accuracer siempre trata de mantener los valores existentes para las columnas modificadas cuando esto es posible. Sin embargo, algunos tipos de conversiones causan perdida de datos - por ejemplo, si quiere convertir una columna string a un entero, todos los valores que no puedan ser convertidos a enteros ser an reemplazos con valor NULL.

La cl usula DROP es usada para remover columnas de una tabla.

Aqu algunos ejemplos:

```
ALTER TABLE Test DROP (Numeric);
```

```
ALTER TABLE Test ADD (NewField WideString(500));
```

```
ALTER TABLE Test DROP CONSTRAINT PK CASCADE
```

```
ALTER TABLE Emp ADD FOREIGN KEY FKDeptID (DeptID) REFERENCES Dept MATCH FULL  
ON DELETE CASCADE ON UPDATE SET DEFAULT
```

1.7.15 Declaración DROP TABLE

Introduction

La declaración DROP TABLE es usada para eliminar una tabla de una base de datos.

Sintaxis

```
DROP TABLE table_name [CASCADE | RESTRICT]
```

La opción CASCADE obliga a eliminar todos los objetos en la base de datos que hagan referencia a esta table (como foreign key constraints.)

Aquí un ejemplo:

```
DROP TABLE Test
```

Nota:

si la tabla no existe, DROP TABLE mostrará una excepción.

1.7.16 Declaración CREATE INDEX

Introduction

La declaración CREATE INDEX es usada para crear nuevos índices a una tabla.

Sintaxis

```
CREATE [UNIQUE] INDEX [IF NOT EXISTS] index_name ON table_name  
(  
  field_name [ASC | DESC] [CASE | NOCASE]  
  [,field_name...]  
)
```

Use CREATE INDEX para crear índices a una tabla. Índices son usados para incrementar la velocidad de búsqueda y clasificación. Índices disminuyen el rendimiento durante la inserción, actualización y eliminación de datos.

La opción UNIQUE indica la restricción sobre la inserción de nuevas líneas con valores de columnas duplicados. De esta manera todas las líneas en una tabla tendrían una combinación única de los valores de las columnas indexadas.

La opción IF NOT EXISTS indica que el índice deberá ser creado solo si no existe para la tabla especificada.

La opción ASC indica el orden ascendente, la opción DESC indica el orden descendente, El valor por defecto es ASC.

La opción CASE especifica una discriminación entre mayúsculas-minúsculas, la opción NOCASE indica que no se tome en cuenta esto. El valor por defecto es CASE.

Ejemplos:

```
CREATE UNIQUE INDEX Text_Index ON Test
(
  Text DESC NOCASE,
  ID
)
```

```
CREATE UNIQUE INDEX Text_Index ON MEMORY [Test MEMORY TABLE]
(
  [My Text] DESC NOCASE,
  ID
)
```

Nota:

Nombres de tablas, columnas e índices pueden ser declarados encerrados en corchetes ([]). De esta manera se pueden usar palabras reservadas (como TABLE) y símbolos especiales (como ' ') en el nombre de la tabla, columnas e índices.

1.7.17 Declaración DROP INDEX

Introduction

La declaración DROP INDEX es usada para eliminar un índice de una tabla..

Sintaxis

```
DROP INDEX [IF EXISTS] table_name.index_name
```

Si la opción IF EXISTS es indicada el índice deberá ser eliminado solamente si existe para esta tabla.

Aquí un ejemplo:

```
DROP INDEX Test.Text_Index
```

1.7.18 Declaración START TRANSACTION

Introduction

La declaración START TRANSACTION es usada para iniciar una transacción en la base de datos.

Para mayor información lea el típico Transacciones en la guía del desarrollador..

Sintaxis

```
START TRANSACTION
```

Aquí está un ejemplo:

```
START TRANSACTION;
INSERT INTO Table1 (NAME) VALUES('aaa');
```

```
UPDATE Table2 SET Field1 = Field1 + 1;  
INSERT INTO Table1 (NAME) VALUES('bb');  
COMMIT;
```

1.7.19 Declaración COMMIT

Introduction

La declaración COMMIT es usada para finalizar la transacción actual y escribe todos los cambios a el archivo de la base de datos.

La opción NO FLUSH indica que el archivo de buffers no debería ser vaciado después de que el commit ha finalizado.

Si esta opción es indicada el COMMIT trabajara un poco mas rápido y todos los cambios serían pasados a la base de datos posteriormente, por un proceso separado de el OS. Si esta opción no es indicada el COMMIT guardara todos los cambios a la base de datos y vaciaría el archivo de buffers, luego de ejecutado todos los datos serían guardados a la base de datos.

Para mayor información lea el capítulo Transacciones en la Guía del Desarrollador.

Sintaxis

```
COMMIT [NOFLUSH]
```

Aquí está un ejemplo:

```
START TRANSACTION;  
INSERT INTO Table1 (NAME) VALUES('aaa');  
UPDATE Table2 SET Field1 = Field1 + 1;  
INSERT INTO Table1 (NAME) VALUES('bb');  
COMMIT NOFLUSH;
```

1.7.20 Declaración ROLLBACK

Introduction

La declaración ROLLBACK es usada para abortar la transacción en progreso y deshacer todos los cambios realizados por esta transacción.

Para mayor información lea el capítulo Transacciones en la Guía del Desarrollador.

Sintaxis

```
ROLLBACK
```

Aquí está un ejemplo:

```
ROLLBACK;
```

1.8 Mecanismo de bloqueo y transacciones, Multi-Usuario y Multi-Tarea

1.8.1 Soporte Multi-Usuario y Multi-Tarea

Introducción

Accuracer puede ser usado en ambos ambientes Multi-Usuario y/o Multi-Tarea así como en un modo mono usuario.

La única restricción es que la edición **SU** puede ser usada solamente en un ambiente Multi-Tarea, pero no en multi-usuario.

La versión **Trial** permite un máximo de 2 conexiones multi-usuario concurrentes y hasta 5 tareas por base de datos abierta en modo Exclusivo.

El acceso Multi-Tarea es el caso donde varias tareas son iniciadas en un mismo proceso y dos o más de ellas puede modificar la misma tabla simultáneamente. Esta situación produce pérdida en las modificaciones o corrupción de datos en el sistema de base de datos que no son seguros en procesos Multi-Tareas. Todas las versiones y ediciones de Accuracer son seguros en procesos Multi-Tarea. Vea el demo Multi-Thread como un ejemplo.

El acceso multi-usuario es cuando algunas diferentes aplicaciones o múltiples instancias de la misma aplicación modifican la misma tabla simultáneamente. No existe diferencia si estas aplicaciones son ejecutadas desde diferentes máquinas o si ellas han sido iniciadas en el mismo computador - en algunos casos esto es un acceso multi-tarea a la base de datos. Esta situación es similar al manejo de multi-tarea si el sistema de base de datos no es multi-usuario habrá pérdida de datos o corrupción de la base de datos. Las ediciones de Accuracer que no soportan acceso multi-usuario son **SU Std** y **SU Pro**. Cuando use la edición **SU** es altamente recomendado que abra la base de datos en modo exclusivo (establezca la propiedad Exclusive del componente TACRDatabase a True). Esto no influenciará el soporte multi-tarea, pero evitará que la base de datos se corrompa por otras aplicaciones. Todas las otras versiones, incluyendo Trial soportan el acceso multi-usuario.

Cada tarea o aplicación deberá usar una única **Session** para el correcto desenvolvimiento. Las sesiones en Accuracer son identificadas por un único valor **SessionID** que toma cada componente TACRQuery o TACRTable que conecta a la base de datos. Cada componente TACRSession o TACRDatabase toma un valor **SessionID** cuando conecta a la base de datos la primera vez. Todos los componentes TACRTable y TACRQuery son conectados a el correspondiente componente TACRSession o TACRDatabase que usan este valor. Todas las modificaciones realizadas por transacciones dentro de la sesión están visibles solamente a los componentes de esta sesión hasta el commit. Cada sesión puede contener solo una transacción activa.

Usando Accuracer en un ambiente Multi-Tarea.

Estas son dos métodos principales de uso correcto de Accuracer en ambientes multi-tareas:

1) Cree un componente TACRSession en cada tarea y use el mismo componente TACRDatabase para todas las tareas.

1) Cree un componente TACRDatabase en cada tarea y use esta para todos los componentes TACRQuery y TACRTable de todas las tareas.

Vea el demo Multi-Thread como un ejemplo.

Usando Accuracer en un ambiente Multi-Usuario.

Esto no necesita de ninguna configuración. Accuracer por defecto está listo para el uso en

ambientes Multi-usuarios.

Lea el típico Mecanismo de Bloqueo para el mejor entendimiento de el modelo de bloqueo utilizado en Accuracer.

Puede ejecutar dos instancias de el utilitario ACRManager o SQLConsole y abra la misma tabla en ambas a ver como esto trabaja.

Nota: No trate de usar la versión SU en modo multi-usuario !!!

Optimizando el rendimiento.

Use acceso exclusivo siempre que esto sea posible. Si necesita solamente acceso multi-tarea a la base de datos, pero que no necesite el acceso multi-usuario. Establezca la propiedad Exclusive de el componente TACRDatabase a True antes de conectar a la base de datos.

En este caso todos los bloqueos serán colocados en memoria, sin tocar ningún byte en el archivo físico de la base de datos. Bloqueos podrán ser chequeados mucho más rápido en este caso.

Si alguna tabla puede ser usada en modo exclusivo, establezca la propiedad del componente TACRTable a True antes de abrirla. Esto acelerará esencialmente el acceso a los registros.

Use transacciones para acelerar la lectura de datos y modificaciones.

1.8.2 Mecanismo de Bloqueo

Introducción

Accuracer puede ser usado en ambos ambientes multi-user y multi-tarea.

El mecanismo de bloqueo es usado para la sincronización entre diferentes usuarios y tareas que modifican los mismos objetos de la base de datos simultáneamente. El bloqueo protege los datos de las lecturas o modificaciones de otros usuarios o tareas.

Los siguientes son los objetos bloqueados en Accuracer;

- Objetos de bajo-nivel de la base de datos (Manejador de espacio libre y lista de tablas) - usadas internamente por el motor de datos Accuracer.
- Objetos de alto-nivel de la base de datos (Tablas y Registros) - usados por ambos el motor de datos Accuracer y el usuario final-

El manejador de espacio libre es un sistema que agrega o elimina páginas (par uso posterior) a el archivo de la base de datos.

La lista de tablas es un sistema que manipula tablas almacenadas dentro de el archivo de base de datos - permite encontrar, crear, borrar y renombrar tablas.

El Mecanismo de Bloqueo (Manejador de Bloqueos) implementado en Accuracer realiza todos lo requerido para bloqueos automáticos. Estos son dos modelos de el manejador de Bloqueos: InMemory y el modo Disk.

El modo InMemory es usado cuando el archivo de la base de datos es abierta en forma exclusiva. esto previene modificaciones por aplicaciones de otros usuarios.

Cada sesión conectada a el archivo de la base de datos toma un identificador único llamado SessionID. Diferentes sesiones son tratadas como diferentes usuarios, también las modificaciones realizadas por una transacción en una sesión no puede ser vista por otras sesiones.

También cada sesión antes de bloquear una tabla o registro revisa el tipo de bloqueo para la compatibilidad con el nuevo bloqueo a establecer por otras sesiones.

La versión mono usuario de Accuracer siempre usa modo InMemory. El manejador de bloqueos en modo InMemory no hace un bloqueo de bytes en el archivo de la base de datos y trabaja mucho más rápido que el modo Disk.

El modo Disk esta dise ñado para acceso multi-usuario a el archivo de la base de datos (por diferentes aplicaciones) en este modo el Manejador de Bloqueos realiza bloqueos y desbloqueos de bytes f ísicos en el archivo de la base de datos. Estas son paginas reservadas por la base de datos y para cada tabla para hacer los bloqueos. El espacio requerido por el Manejador de Bloqueos es 1 byte por cada conecci ña a el archivo de la base de datos y 11 bytes por cada conecci ña a cada tabla. El m áximo n úmero de conecciones (TACRDatabase.Opci ñn.MaxSessionCount) influencia ambos tama ños de el archivo de la base de datos y el rendimiento de el acceso multi-usuario a el archivo. Si establece un gran n úmero de maximas concurrentes conecciones cada operaci ñn de bloqueo trabajara mas lenta y vice versa. Este aprovechamiento permite obtener un gran rendimiento para la validaci ñn de bloqueos y uso de el motor de datos Accuracer en diferentes plataformas. Mas de todos los sistemas operativos modernos bloquean y desbloquean bytes dentro del archivo de la base de datos, mientras otras t écnicas pueden tener problemas bajo diferentes plataformas. Por ejemplo, el bloqueo de bytes fuera del archivo trabaja en todas las versiones de MS Windows, pero nunca funcionara bajo Unix, Linux y Novell.

Modos de bloqueo

Los siguientes son los tipos de bloqueos en Accuracer:

- IS - usado para abrir la tabla en modo compartido y permanecer abierta hasta ser cerrada.
- X - usado para abrir la tabla en modo exclusivo y permanecer abierta hasta ser cerrada.
- S - usado para bloquear la tabla cortamente en el modo solo lectura.
- IRW - usado para bloqueo continuo de la tabla en el modo solo lectura.
- RW - usado para bloqueos cortos de la tabla en modo exclusivo.
- U - usado por bloqueos cortos o continuos de registros (operaciones de Edici ñn o borrado en TACRDataset).

Cada sesi ñn puede bloquear solamente un registro en la tabla. Si una sesi ñn bloquea un registro (modo U), otras sesiones no pueden bloquear este registro y no pueden editarlo o actualizarlo. Sin embargo ellos pueden leer estos registros.

Antes de bloquear una tabla o registro el Manejador de Bloqueos revisa nuevos tipos de bloqueo de otras sesiones para la compatibilidad de el bloqueo que se va a establecer. Si el bloqueo falla esta sesi ñn esperara un tiempo especificado en TACRDatabase.LockParams y tratara de colocar el bloqueo nuevamente.

Si el n úmero de intentos no logrados excede el valor indicado en TACRDatabase.LockParams.RetryCount una excepci ñn es mostrada.

Tabla de compatibilidad de esquemas de bloqueos:

Modo de Bloqueo	X	IS	S	IRW	RW
X	NO	NO	NO	NO	NO
IS	NO	SI	SI	SI	SI
S	NO	SI	SI	SI	NO
IRW	NO	SI	SI	NO	NO
RW	NO	SI	NO	NO	NO

Optimizando el rendimiento

Estos son tres m étodos para mejorar el rendimiento de acceso mlulti-usuario:

- 1) Use transacciones.
 - 2) Abra las tablas en modo exclusivo para operaciones críticas.
 - 3) Combine los métodos 1 y 2
- Si no necesita acceso multi-user puede también abrir la base de datos contenedora en modo exclusivo.

1.8.3 Transacciones

Introducción

Transaction is the logical sequence of the database modification operations that can be treated as an atomic unit of work. Transactions have the following properties (ACID):

Transacciones es la secuencia lógica de las operaciones de modificaciones de la base de datos que pueden ser tratadas como una unidad de trabajo atómica. Las transacciones tienen las siguientes propiedades (ACID):

Atomicidad

Una transacción permite el agrupamiento de uno o más cambios a líneas de tablas dentro de una base de datos para formar una atómica o indivisible operación. Esto quiere decir que todos los cambios se hacen o no se hacen. Si por alguna razón no puede ser completada, todo lo cambiado puede ser restaurado a su estado previo antes del inicio de la transacción vía una operación rollback.

Consistencia

Las transacciones siempre operan sobre una vista consistente de los datos y cuando ellas finalizan entregan los datos en un estado consistente. Los datos se puede decir para ser consistentes así como la conformidad a un conjunto de imprevistos, como que dos líneas en la tabla clientes no pueden tener el mismo identificador de cliente y todas las ordenes tienen asociada la línea de clientes. Mientras la transacción ejecuta estos imprevistos pueden ser violados, pero ninguna otra transacción tendrá permitido ver estas inconsistencias, y todas las inconsistencias tendrán que ser eliminadas en el momento en que la transacción finaliza.

Aislamiento

A una transacción dada, debe aparecer como si esta ejecutándose por sí misma sobre la base de datos. Los efectos de concurrencia de transacciones en ejecución son invisibles durante la ejecución de esta transacción, y los efectos de estas transacciones son invisibles a otras transacciones hasta que la transacción sea aceptada.

Durabilidad

Una vez que una transacción es aceptada, estos cambios son garantizados incluso en fallos de sistema subsecuentes. Hasta que una transacción no sido aceptada (committed), no solamente los cambios realizados por la transacción no durables, pero son garantizados para no persistir en caso de un fallo de sistema, como recuperaciones de emergencia y todos los efectos serán rechazados

Implementación de transacciones en Accuracer

Accuracer soporta transacciones solamente para bases de datos de disco. Tablas en-memoria no se ven envueltas en transacciones.

Todos los datos modificados por las transacciones son almacenados en RAM, así si alguna falla ocurre durante el procesamiento de la transacción todas las modificaciones se perderán y la base de datos estará en el mismo estado antes de iniciada la transacción. El archivo de la base de datos puede corromperse solamente si una falla ocurre durante el proceso de commit.

Múltiples transacciones sobre una misma base de datos pueden realizarse simultáneamente solamente si ellas son creadas en diferentes sesiones.

Vea el t plico de soporte Multi-usuario y multi-tarea para entrenarse mas acerca de las sesiones en Accuracer.

Todas las modificaciones realizadas por una transacci n no pueden ser accesadas por otras sesiones hasta que el commit halla finalizado.

Las transacciones pueden finalizar por la culminaci n de un Commit o Rollback, El Commit tratara de escribir todos los cambios realizados por la transacci n a la base de datos y despu s desbloquea todas las tablas involucradas en la transacci n.

El Commit por defecto desaloja el archivo de buffer despu s de escribir los cambios, tambi n toda la data sera guardada en el archivo inmediatamente.

Opcionalmente Commit puede saltar este proceso que trabaja mas r pido que con desalojamiento del buffers

The Rollback discards all changes, removes all pages added during the transaction and after that unlocks all tables involved in the transaction.

El Rollback descarga todos los cambios, remueve todas las paginas a adidas durante la transacci n y despu s desbloquea todas las tablas involucradas en la transacci n.

Todas las tablas en la base de datos abiertas antes de iniciada la transacci n o durante su procesamiento son automaticamente involucradas en la transacci n. De manera que ellas seran bloqueadas en modo S (ver t plico Mecanismo de Bloqueos) y no puede ser modificada por otras sesione.

Si la transacci n modifica algunas tablas esto bloquea estas tablas in modo RW de manera que otras sesiones no pueden iniciar operaciones de inserci n, eliminaci n o edici n sobre esta tabla ni declaraciones SQL como INSERT, UPDATE, DELETE.

Durante el Commit la transacci n intentara establecer bloqueos RW a todas la tablas modificadas, y mostrara una excepci n si falla, otras sesiones no podr n leer registros provenientes de estas tablas.

Nivel de aislamiento

El nico nivel en Accuracer es READ COMMITTED. De manera que todos los cambios realizados por otras transacciones no podr an ser vistas por otras sesiones hasta que el Commit halla finalizado.

Ejecutando una transacci n

Una transacci n puede ser ejecutada de dos maneras:

- 1) Usando el componente TACRDatabase - M todos StartTransaction, Commit, Rollback
- 2) Ejecutando declaraciones SQL START TRANSACTION, COMMIT, ROLLBACK

El demo de transacciones muestras ambos m todos.

No olvide acerca de la manipulaci n de excepciones durante el procesamiento de una transacci n: Ud., deberia ejecutar Rollback manualmente si una excepci n es disparada. Excepciones pueden ser causadas por la imposibilidad de bloquear alguna tabla o por razones de violaciones como constraints.

Como una transacci n incrementa el rendimiento

La transacci n bloquea todas las tablas abiertas por la actual sesi n y mantiene estos bloqueos hasta que Commit o Rollback sean llamados. Cada tabla involucrada en la transacci n no puede ser modificada por otras sesiones, as no es necesario releer datos desde la base de datos y todos los cambios son salvados solo durante este proceso de Commit, no despues de cada sencilla operaci n.

El máximo rendimiento puede ser obtenido abriendo las tablas en modo exclusivo y ejecutando una transacción. Incluso las lecturas registra trabajos mucho más rápidos dentro de una transacción.

Operaciones incompatibles con transacciones

Todas las operaciones que requieran un acceso exclusivo a la tabla o la base de datos no puede hacerse cuando una transacción ya ha sido iniciada. Sin embargo, todas las tablas que no están involucradas en la transacción no puede ser accesadas en modo exclusivo.

Aquí esta una lista de las operaciones que son incompatibles con transacciones:

- Reparación, Compactación o configuración Changing de la base de datos.
- Vaciar, Reestructurar, Borrar y Renombrar tablas envueltas en transacciones.
- Crear y borrar índices sobre tablas envueltas en transacciones.

1.9 Motor Cliente-Servidor

1.9.1 Introducción

Introducción

Accuracer soporta ambas tecnologías Cliente-Servidor y Archivo-Servidor (Multi-user).

Cuando use el modo Archivo-Servidor la base de datos es indicada por la propiedad DatabaseFileName del componente ACRDatabase (la propiedad LocalDatabase debería estar establecida a True). En este caso cada aplicación cliente lee y escribe a la base de datos directamente, usando la sincronización de bloqueos del OS.

Cuando usa el modo Cliente-Servidor la base de datos es indicada por la propiedad ConnectionParams de el componente TACRDatabase (la propiedad LocalDatabase debería estar establecida a False). Aplicaciones clientes envían requerimientos a la base de datos servidora vía red local usando un protocolo basado en UDP. La aplicación servidora ejecuta el requerimiento del cliente y envía la replica a este, accediendo la base de datos en modo exclusivo (si OpenDatabasesInExclusiveMode esta establecida a true) o como un archivo-servidor.

Como usar

1) Usando un base de datos servidora

Puede usar el servidor de base de datos Accuracer como un servicio Windows NT/XP como una aplicación Windows.

Para instalar el servidor como un servicio debería ejecutar esto con la opción /install
c:\Accuracer\Utils\Bin\ACRServer\AccuracerDatabaseServer.exe /install

Después puede iniciar o detener este como un servicio.

Para desinstalar un servicio ejecutándose en el servidor user la opción /Uninstall
c:\Accuracer\Utils\Bin\ACRServer\AccuracerDatabaseServer.exe /uninstall

También puede ejecutar este como una aplicación típica de Windows

c:\Accuracer\Utils\Bin\ACRServer\AccuracerDatabaseServer.exe

En este caso puede usar un menú popup en el tray icono para iniciar o detener el servidor.

La configuración por defecto son aplicadas cuando la aplicación es iniciada (establezca Active a True) si no esta en el archivo de configuración.

En este caso el servidor creará el archivo de configuración con valores por defecto durante el primer inicio.

El servidor provee acceso solamente a las bases de datos indicadas en las propiedades DatabaseNames y DatabaseFileNames.

Valores por defecto para estas propiedades apuntan a base de datos Demos\Data\DBDemos.adb

El puerto del servidor puede ser establecido via la propiedad LocalPort de TACERServer.

Para trabajar con clientes desde máquinas remotas debe establecer el dirección IP de el servidor en el parámetro **LocalHost** de el archivo de configuración. Valor por defecto 'LocalHost' permite trabajar solo con clientes iniciados en la misma máquina.

2) Conectando a la base de datos en la base de datos servidora.

Puede usar un componente TACRDatabase para conectar la base de datos remota.

Establezca la propiedad LocalDatabase a False, indique los necesarios parámetros de conexión en la propiedad ConnectionParams (DatabaseName, RemoteHost, RemotePort y LocalPort) y abrir la base de datos (Open o Active =true).

Valores por defecto permiten probar localmente.

DatabaseName = DBDemos

RemoteHost = localhost

RemotePort = 6669

LocalPort = 6668

Vea el demo Cliente como un ejemplo de como conectar a una base de datos remota.

Si OpenDatabasesInExclusiveMode esta establecida a true el máximo número de conexiones a una simple base de datos es:

- 5 en la versión trial;
- 2^{31} en la versión comercial;

Si OpenDatabasesInExclusiveMode esta establecida a false el número máximo de conexiones a una simple base de datos es:

- 2 en la versión Trial;
- El parámetro MaxSessionsCount de la base de datos en la versión comercial;

Características Avanzadas

El servidor de base de datos Accuracer le brinda la posibilidad de cambiar cualquier query SQL antes de la ejecución o abortar.

Esto puede hacer su aplicación más flexible:

- Puede cambiar la definición de datos sin recompilar y reinstalar las aplicaciones clientes.
- Puede bloquear algunos queries SQL por razones de seguridad.
- Puede grabar todos los queries SQL ejecutados por los clientes.

Para mayor información lea la descripción del evento TACRServer.OnSQL en Accuracer Referencia de Componentes.

Otra gran ventaja del Servidor Accuracer es que soporta mensajes propios de su aplicación.

Ahora puede establecer alguna comunicación entre el servidor y algún cliente conectado a este.

Puede enviar y recibir texto, binarios y mensajes stream desde ambos lados: cliente y servidor.

Puede comunicar algún cliente con otro cliente (atraves del servidor), cliente con servidor y servidor con cliente.

Y estos mensajes pueden ser enviados y recibidos en cualquier momento, simultaneamente con acceso a las tablas y ejecutando scripts SQL.

Para mayor información revise los eventos OnReceiveTextMessage, OnReceiveBinaryMessage, OnReceiveStreamMessage y el método SendMessage de los componentes TACRDatabase y TACRServer en Accuracer Referencia de Componentes.

Accuracer can compress and / or encrypt network traffic (ConnectionParams property of TACRDatabase and CryptoParams property of TACRServer).
Accuracer puede comprimir y/o encriptar el tráfico de red (propiedad ConnectionParams de TACRDatabase y la propiedad CryptoParams de TACRServer).

Limitaciones

las siguientes son limitaciones de la versión actual Cliente-Servidor;

- RepairTable, FlushFileBuffers, GetLastAutoinc y SetLastAutoinc no están soportadas
- OnFilterRecord no trabaja con tablas remotas y en queries remotos en vivo.

Nosotros trataremos de eliminar la mayoría de estos en la próxima versión (excluyendo OnRecordFilter).

1.10 Migración

1.10.1 Sobrevista

Introducción

Ahora hay una cantidad de sistemas de bases de datos para diferentes plataformas en el mundo. En la mayoría de los casos tratamos de usar nuestra base de datos favorita para nuestros proyectos, pero algunas veces hacemos frente que esto no puede satisfacer nuestras necesidades más. Estas son cantidades de razones por las que tenemos carencia de características, soporte técnico o documentación, costos irrazonables, bajo rendimiento, demasiado uso de recursos (RAM, CPU, espacio en disco o un enorme número de archivos, fallos de seguridad, movimientos a otra plataforma, etc.

Este es un buen momento para migrar a otros sistemas de base de datos como nuestro **Accuracer**

Como comenzar

Estos son dos pasos principales para la migración:

- 1) Mover datos a Accuracer
- 2) Actualizar proyectos y aplicaciones

Nosotros recomendamos comenzar primero con una, asegurarse de estar habilitado para probar la nueva base de datos, examinar el rendimiento, revisar la compactación, estar seguro que todo lo requerido está funcionalmente soportado, vea las nuevas características determine que puede ser usadas ahora antes de comenzar la más compleja parte de este trabajo - paso 2

Nosotros sabemos que hay una cantidad de sistemas de bases de datos y ambientes de desarrollo, así nosotros dividimos esto en tres grupos separados

- 1) BDE o ODBC
- 2) EasyTable
- 3) Otros sistemas de base de datos y plataformas

Nota: Si tiene algún problema durante el proceso de migración, contacte nuestro Equipo de soporte técnico - Nosotros le ayudaremos tanto como nos sea posible.

1.10.2 Migración desde BDE

Introducción

EL proceso de migración puede ser muy simple:

- 1) Convertir sus datos con el utilitario DBTransfer.
- 2) Reemplace sus componentes TTable, TQuery, TSession y TDatabase a componentes TACRTable, TACRQuery, TACRSession y TACRDatabase
- 3) Establezca la propiedad DatabaseFileName del TACRDatabase a el camino a el archivo de la base de datos instado de Alias / Directorio.
- 4) Establezca la propiedad Exclusive de el componente TACRTable a el mismo valor como en el componente TTable
- 5) Si necesita acceder a el archivo de base de datos solamente desde una única instancia de la aplicación (acceso mono-usuario o multi-tarea) establezca la propiedad Exclusive de TACRDatabase a True.
- 6) Remueva la unidad DBTables de sus unidades. De otra manera su proyecto seguirá requiriendo BDE.

No es necesaria ninguna otra configuración.

1.10.3 Migración desde EasyTable

Introducción

El proceso de migración es muy simple:

- 1) Convierta sus datos con el utilitario de conversión
 - 2) Reemplace sus componentes TEasyTable, TEasyQuery, TEasySession y TEasyDatabase por TACRQuery, TACRSession y TACRDatabase
 - 3) Establezca la propiedad DatabaseFileName del componente TACRDatabase con el sitio donde está el archivo de base de datos Accuracer.
 - 4) Establezca la propiedad Exclusive del componente TACRTable a True para tablas que deberán ser accesadas exclusivamente.
 - 5) Si necesita acceder a la base de datos desde una instancia solamente de la aplicación (acceso mono-usuario o multi-usuario), establezca la propiedad Exclusive de TACRDatabase a True.
 - 6) Elimine las unidades EasyTable y Etbl* de la cláusula uses de sus unidades. De otra manera su proyecto requerirá EasyTable.
- Esto no necesitará algunas otras configuraciones.

1.10.4 Migración desde otros sistemas de base de datos y plataformas

Introducción

Lea el capítulo Importar y Exportar para conocer más de cómo puede mover sus datos a Accuracer. Si su proyecto usa base de datos que proveen componentes análogos a TTable, TQuery, TSession and TDatabase entonces el proceso de migración es similar a la migración desde el BDE. Reemplazar componentes ADO, Interbase o DBExpress es casi la misma tarea.

En otros casos tendrá que reescribir todo su código para usar los componentes Accuracer.

Nota: Si tiene algún problema durante el proceso de migración, contacte nuestro Equipo de Soporte Técnico - nosotros le ayudaremos tanto como sea posible.

1.10.5 Importar y Exportar

Introducción

Estos son dos métodos para importar data a Accuracer:

- 1) Usando el utilitario Accuracer DBTransfer o Convert.
- 2) Haciendo su propio convertidor.

El primer método puede ser usado si tiene una base de datos que pueda ser accesada via BDE o ODBC (como Paradox, Interbase, Access, DBase, FoxPro, Oracle y SQLServer). De otra modo puede hacer su propio convertidor.

Importando tablas desde el BDE o ODBC

Si necesita mover datos entre la base de datos que puede ser accesada via BDE o ODBC la mejor manera de hacer esto es usando el utilitario DBTransfer localizado en <carpeta de instalación Accuracer>\Utils\Bin\DBTransfer\

Puede seleccionar alguna tabla existente de Paradox o DBase (FoxPro) e importar estas por el utilitario DBTransfer.

También puede crear cualquier BDE o ODBC alias, ejecutar DBTransfer e importar cualquier tabla desde este alias.

Importando tablas desde EasyTable

Solo ejecute el utilitario Convert y seleccione el archivo de base de datos Easy Table existente.

Importando tablas desde MySQL

Ver demo MySQLImport.

Importando tablas desde CSV (valores separados por comas)

Ver demo CSVImport.

Importando tablas desde otros sistemas de bases de datos

Si su base de datos no puede ser accesada via BDE o ODBC, y esta no cuenta con utilitarios de exportación que puedan convertir esta un formato compatible con BDE entonces Ud., tiene que crear su propio convertidor.

Estas son tareas fáciles si puede acceder las tablas en Delphi, C++ Builder o Kylix:

- Crear una nueva aplicación
- Crear los componentes necesarios para acceder su base de datos
- Crear los componentes TACRDatabase y TACRTable, establecer la propiedad Database de ambos componentes al mismo valor, como 'TestDB'
- Establezca las propiedad DatabaseFileName de el componente TACRDatabase a la base de datos existente, como 'C:\test.adb'. Puede crear la base de datos usando el utilitario ACRManager o usando el método CreateDatabase de TACRDatabase
- Cree un nuevo procedimiento de conversión - abra alguna tabla existente en su base de datos con el componente de tabla (derivado de TDataSet). cierre la tabla Accuracer

(TACRTable.Close), establezca el mismo nombre de tabla (TACRTable.TableName) y llame el método de importación de TACRTable con su componente de tabla (TACRTable.ImportTable).
 Vea el código fuente de Convert como un ejemplo.
 <Carpeta de instalación Accuracer>\Utils\Source\Convert\

If your database cannot be accessed by TDataSet descendant component you can try to export it to SQL script or contact our Support Team.

Si su base de datos no puede ser accesada por un descendiente del componente TDataSet puede tratar de exportar este a scripts SQL o contactar a nuestro Equipo de Soporte

Nota: Si tiene algún problema durante el proceso de migración, contacte nuestro Equipo de Soporte Técnico - nosotros le ayudaremos tanto como sea posible.

1.11 Afinamiento y optimización

1.11.1 Revisión

Introduction

Las bases de datos modernas, como Accuracer, proveen cantidad de diferentes configuraciones que pueden ser usados para conseguir mayor rendimiento, haciendo las bases de datos más compactas, usando menos cantidad de RAM, etc. Si no está seguro de que necesita, por favor contacte nuestro Equipo de Soporte.

Usando Indices

Si realiza búsquedas o filtros sobre grandes tablas la forma principal de proveer rendimiento es creando los índices necesarios.

Esto puede hacerse usando la propiedad TACRTable.IndexDefs antes de crear una tabla o ejecutando el método TACRTable.AddIndex o la declaración SQL CREATE INDEX cuando la tabla existe.

Si realiza búsquedas sobre un simple campo ('Name' por ejemplo) entonces debe crear un índice por este campo.

Esto es muy importante.

Es muy importante crear índices sobre campos string (char, varchar, wide char o wide varchar) con la misma sensibilidad a mayúsculas de la búsqueda. Si hace una búsqueda no sensible deber crear un índice no sensible y al contrario si es sensible haga un índice sensible.

Ejemplo 1:

```
TACRTable1.AddIndex('case_ins_index', 'Company', [ixCaseInsensitive]);
TACRTable1.Locate('Company', ['AidAim Software'], [loCaseInsensitive]);
```

Debe crear índices multi-campos si hace búsquedas por múltiples campos.

Ejemplo 2:

```
TACRTable1.AddIndex('complex_index', 'Company;Contact;Phone', []);
TACRTable1.Locate('Company;Contact;Phone', VarArrayOf(['Sight Diver','P']), loPartialKey);
```

Las declaraciones SELECT también se ejecutan más rápido cuando los índices necesarios existen.

Recomendamos crear índices por cada campo a ser usado en declaraciones DISTINCT, WHERE,

ORDER BY y FROM.

Si usa ORDER BY para multiples campos, cree un unico indice sobre todos estos campos como en el ejemplo 2.

Si hace una búsqueda de contexto con un operador LIKE no es necesario crear ning un indice - ya que esto no proveera mayor rendimiento.

Nota: Los indices disminuyen la rapidez en la modificación de tablas especialmente con las opciones Primary o Unique.

Optimizando la Navegación

Puede usar de TACRDataset los métodos LockTable / UnlockTable para optimizar la velocidad en la carga de registros

Ejemplo:

```
ACRTable1.LockTable;
try
  ACRTable1.First;
  while not ACRTable1.Eof do
    begin
      // ... some record processing
      ACRTable1.Next;
    end;
finally
  ACRTable1.UnlockTable;
end;
```

Acceso Exclusivo

Si necesita acceso multi-usuario, establezca la propiedad Exclusive de los componentes TACRDatabase o TACRTable a True. Esto incrementará significativamente el rendimiento, pero otras aplicaciones no estarán habilitadas a para acceder las tablas y bases de datos abiertas exclusivamente.

Máximo Número de Conexiones.

Puede establecer el número máximo de conexiones para cada base de datos (usando la ACRManager or TACRDatabase o la propiedad Options de TACRDatabase) a cualquier valor desde 1 hasta la constante ACRMaxSessionCount. Los valores más pequeños incrementarán el rendimiento en ambientes multi-usuarios y también brindarán una base de datos más compacta.

Como Hacer Su Base De Datos Mas Compacta

Puede hacer su archivo mas compacto. Si usa campos varchar en vez de campos string de longitud fija.

También puede usar compresión para campos BLOB y campos Varchar.

Lea el artículo BLOB y campos Varchar para mayor detalle.

Otro consejo es ejecutar el método TACRDatabase.CompactDatabase periódicamente (o usar el utilitario ARCManager) para remover espacio no usados dentro del archivo de la base de datos.

Puede usar la propiedad Density de el componente TACRDatabase para obtener el porcentaje de espacio usado dentro de el archivo de la base de datos.

Otras configuraciones

Vea el t pico Variables y Constantes en la Guia de Referencia.

Nota: Si tiene alg n problema durante el proceso de migraci n, contacte nuestro Equipo de Soporte T cnico - nosotros le ayudaremos tanto como sea posible.

1.12 Apendice

1.12.1 Diferencias contra BDE

Principales diferencias:

- Supported data types
- M ximos campos por tabla: ~ 2^31
- M ximos indices por tabla: ~ 2^31
- M ximos campos por indice: ~ 2^31
- M ximos caracteres por nombre de campo: 255 characters
- Algoritmos de compresi n BLOB y Varchar: ZLIB, BZIP, PPM
- Secuencias soportadast (AutoInc campos basados en secuencias)
- Reestructuraci n de tablas
- Soporte Unicode
- Sin manejadores externos ni dlls
- Versi n para Kylix esta disponible
- Encriptamiento severo de la base de datos con cantidad de algoritmos y configuraci n.
- Campo ftString es un equivalente del tipo Varchar - campo string de longitud variable.
- Campo ftFixedChar es un equivalente del tipo Char - campo string de longitud fija.

Si necesita informase de cuales caracter sticas necesita antes de todas, por favor contactenos a : Support@aidaim.com.

Si quiere informarse acerca de nuevas entregas de este componente, puede suscribirse a nuestra lista de correos:
<http://www.aidaim.com/info/subscr.php>

1.12.2 Tipos de datos soportados

Accuracer soporta los siguientes tipos de datos en tablas:

<u>Tipo de dato SQL</u>	<u>Descripci n</u>	<u>Correspondiente TFieldType</u>
Char, FixedChar	Campo de caracarteres fijos	ftFixedChar
Varchar, Varchar2, String	Caracter o campo de longitud variable	ftString
WideChar, FixedWideChar	Fixed wide character field	ftWideString
WideVarchar,	Wide character or variable	ftWideString

WideString	length wide string field	
Shortint, SignedInt8	8-bit integer field	ftSmallint
Smallint, SignedInt16	16-bit integer field	ftSmallint
Integer, SignedInt32	32-bit integer field	ftInteger
Largeint, Int64, SignedInt64	64-bit integer field	ftLargeint
Byte, UnsignedInt8	Byte field	ftWord
Word, UnsignedInt16	16-bit unsigned integer field	ftWord
Cardinal, UnsignedInt32	32-bit unsigned integer field	ftLargeint
AutoInc, AutoIncInteger	Auto-incrementing 32-bit integer counter field	ftAutoinc
AutoIncShortint	Auto-incrementing 8-bit integer counter field	ftAutoinc
AutoIncSmallint	Auto-incrementing 16-bit integer counter field	ftAutoinc
AutoIncLargeint	Auto-incrementing 64-bit integer counter field	ftAutoinc
AutoIncByte	Auto-incrementing byte counter field	ftAutoinc
AutoIncWord	Auto-incrementing 16-bit unsigned integer counter field	ftAutoinc
AutoIncCardinal	Auto-incrementing 32-bit unsigned integer counter field	ftAutoinc
Single	Single floating-point numeric field	ftFloat
Float, Double	Double floating-point numeric field	ftFloat
Extended	Extended floating-point numeric field	ftFloat
Boolean, Logical, Bool, Bit	Boolean field	ftBoolean
Currency, Money	Money field	ftCurrency
Date	Date field	ftDate
Time	Time field	ftTime
DateTime	Date and time field	ftDateTime
TimeStamp	Date and time field accessed through dbExpress	ftTimeStamp
Bytes	Fixed number of bytes (binary storage)	ftBytes
VarBytes	Variable number of bytes (binary storage)	ftVarBytes
Blob	Binary Large Object field	ftBlob
Graphic	Bitmap field	ftGraphic
Memo, Clob	Text memo field	ftMemo
FormattedMemo, FmtMemo	Formatted text memo field	ftFmtMemo
WideMemo, WideClob	Unicode text memo field	ftMemo

1.12.3 Internacioanlización y regionalización

Introducción

Este artículo discute una guía para crear aplicaciones que puede distribuir en el mercado internacional. A continuación lo que debe planear. debe reducir la cantidad de código y tiempo necesario para hacer su aplicación opere bien en estos mercados foráneos, así como en el mercado doméstico.

Algunas cosas debe saber para crear aplicaciones internacionales con Accuracer.

Moneda, punto flotante y formato fecha/hora

Algunas veces tendrá que convertir moneda, Fecha/Hora o punto flotante a valores string. Por ejemplo si necesita establecer la propiedad Filter de Accuracer para la condición 'Birthday=01/01/1970'

Accuracer siempre usa los valores actuales de DataSeparator, TimeSeparator y DecimalSeparator.

También debe usar las funciones DateToStr/TimeToStr o FloatToStr para obtener valores fecha/hora o float convertidos.

Locale and strings sort order

Puede usar las propiedades IndexName o IndexFieldNames para establecer el índice activo, y consecuentemente, ordenar la tabla activa basada en la definición del índice seleccionado para el orden.

Sin embargo clasificar por string depende de la región de actual system/user y es específico para varios lenguajes.

Accuracer usa operaciones específicas regionales, también si usa el tipo de dato ftString entonces todos los registros serán clasificados por este campo y serán ordenados usando la configuración regional actual.

Si quiere soporte para lenguajes Asiáticos debe usar el conjunto de caracteres Unicode.

Soporte Unicode

En el conjunto de caracteres Unicode, cada carácter es representado por dos bytes, así un string Unicode es una secuencia de palabras de dos-byte, no bytes individuales. Caracteres Unicode y strings son también llamados wide characters y wide character strings. Los primeros 256 caracteres Unicode mapean el conjunto de caracteres ANSI.

Accuracer implementa soporte Unicode a través del tipo de dato ftWideString.

A continuación ejemplos de como establecer y obtener un campo Unicode.

```
var ws: WideString;
with MyAccuracer do
begin
  // get data from Unicode field
  ws := FieldByName('Unicode').Value;
  // do something with data in ws
  ws := 'Ejemplo string';
  // set data to Unicode field
  Insert;
  FieldByName('Unicode').Value := ws;
```

```
Post;  
end;
```

Campos WideMemo pueden ser accedidos usando los métodos TACRDataset, SetWideMemoField y GetWideMemoField:

```
var ws: WideString;  
with MyAccuracer do  
begin  
  // do something with data in ws  
  ws := 'Ejemplo string';  
  // set data to Unicode field  
  Insert;  
  SetWideMemoField(FieldByName('Unicode'), ws);  
  Post;  
  // get data from Unicode field  
  ws := GetWideMemoField(FieldByName('Unicode'));  
end;
```

Nota: Campos Wide string pueden ser usados solamente con versiones que soporten la función CompareStringW (NT, 2000, XP, 2003). Windows 9x, Me no soportan Wide Strings.

1.12.4 Limitaciones

- Campo tipo WideString no son soportados en Delphi 4, C++ Builder 4
- Campo SQLTimeStamp no es soportado en Delphi 4,5, C++ Builder 4,5
- Campos BCD no son soportados.
- Trial Versión puede ejecutar declaraciones SELECT SQL solamente.
- Máximo número de conexiones multi-usuarios en Trial Versión es 2
- Máximo número de conexiones multi-tareas en mono-usuario (modo Exclusivo) en Trial Versión es 5
- Versión SU Std puede ser usada solo en ambiente mono-usuario o en múltiples tareas dentro de una misma instancia de su aplicación.
- OnFilterRecord no trabaja con tablas remotas y queries en vivo.
- RepairTable, GetLastAutoinc y SetLastAutoinc no está soportado en bases de datos remotas.

Index

- A -

ABS Function 41
Access 72
ADO 72
Aggregate Functions 29
ALTER TABLE Statement 59
AVG Function 29

- B -

BDE 72
BLOB and Varchar fields 15
BLOB Compression 15
BLOB fields use 14

- C -

CAST Function 48
CEILING Function 42
Client-Server 69
Commit 67
COMMIT Statement 63
Compactness 74
Compression 15
Connections 64
Contents 8
COUNT Function 29
CREATE INDEX Statement 61
CREATE TABLE Statement 56, 57
Creating a table 6
CUMPROD Function 43
CUMSUM Function 42
CURRENT_DATE Function 32
CURRENT_TIME Function 32
CURRENT_TIMESTAMP Function 33

- D -

Data 71, 73
Database 73

Date and Time Functions 31
DAY Function 33
DAYNAME Function 33
DAYOFWEEK Function 33
DBase 72
DELETE Statement 55
Differences from TTable 76
DISTINCT 49
DROP DATABASE Statement 56
DROP INDEX Statement 62
DROP TABLE Statement 61

- E -

EasyTable 72
EXCEPT 49
Exclusive access 64
Export 71, 73
EXTRACT Function 34

- F -

Features 2
Filtering tables 11
FLOOR Function 42
FoxPro 72
ftFixedChar 15
ftString 15
Functions 28

- G -

Getting Help from Technical Support 5
GROUP BY 49
GROUP_CONCAT Function 30

- H -

HEX constants 28
HEX Function 44
HOUR Function 34
How to Buy 5

- I -

Import 71, 73

InMemory tables 6
 INSERT Statement 54
 Internationalization and localization 78
 Introduction 2
 ISNULL Function 40

- J -

JOIN 49

- L -

LASTAUTOINC Function 40
 LENGTH Function 45
 Locks 65
 LOWER Function 46
 LTRIM Function 46

- M -

Mathematical Functions 40
 MAX Function 31
 Migration 71
 MIN Function 30
 MINUS 49
 MINUTE Function 35
 Miscellaneous Functions 39
 MOD Function 42
 MONTH Function 35
 MONTHNAME Function 35
 Moving 73
 Moving Data 71
 MSECOND Function 35
 Multi-Thread 64
 Multi-User 64

- N -

Naming conventions 19
 Navigating Tables 9
 Network 69
 NOW Function 33

- O -

ODBC 72

Operators 26
 ORDER BY 49
 Other Functions 39
 Overview 18

- P -

Paradox 72
 Parameters 25
 Performance 74
 POS Function 46
 POWER Function 44

- Q -

QUARTER Function 36

- R -

RAND Function 44
 RANDOM Function 44
 Record locks 65
 Restructuring a table 17
 Rollback 67
 ROLLBACK Statement 63
 ROUND Function 43
 RTRIM Function 47

- S -

SECOND Function 36
 SELECT Statement 49
 Sessions 64
 Setting up a table component 6
 SIGN Function 41
 Sorting records 13
 Speed 74
 SQL 18
 SQL Functions 28
 Start transaction 67
 START TRANSACTION Statement 62
 String Functions 45
 SUBSTRING Function 47
 SUM Function 31
 Supported data types 76
 SYSDATE Function 33

- T -

Table locks 65
Tables 73
Third Party DB systems 72
TOBLOB Function 49
TODATE Function 36
TOP 49
TOSTRING Function 38
Transactions 67
TRIM Function 47
TRUNCATE Function 43
Tuning and Optimizations 74
Type Conversion Functions 48

- U -

Unicode 78
UNION 49
UPDATE Statement 54
UPPER Function 47
Using parameters 25

- V -

Varchar 15
Varchar Compression 15

- W -

WEEKDAY Function 39
WideMemo 78
WideString 78

- Y -

YEAR Function 39

Endnotes 2... (after index)

Back Cover